programmers journal

The Resource Journal for IBM PC Programmers

Office Copy

Volume 1 Number 6

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROGRAMMER'S JOURNAL

CONTENTS

HOUSEKEEPING 2

LETTERS 3

PJ PROFILE: Interview with Dan Rollins 6
Books and software, the best of both

PJ FORUM: STRUCTURED PROGRAMMING REVISITED / Paul A. Barkley 11
Bridging academic theories and real applications

ASM: PRINTER CONFIGURATION IN ASSEMBLY / Rick Millhollin 17
Straight forward assembly programming

DOS: THE DOS PATCH UTILITY / Brett Salter 20
The easy way to update your system

C: C THE DIFFERENCE / Ed Keating 24
C and BASIC compared, module by module

ASM: FLYING SAUCERS / Dan Rollins 27
High speed assembly graphics modules

LEGAL: HIGHPOINTS OF SOFTWARE PROTECTION / Robert Dunaway 33
Who really owns your software?

PASCAL: DOS CALLS IN PASCAL / from Peeks n' Pokes 34
Function calls from Pascal programs

ASM: DISK PARAMETER EQUATES / Morton F. Kaplon 36
Using equates for flexibility

BASIC: ARRAY AND FILE HANDLING / Greg Estes 37
Where should you store your information

dBASE II: MARATHON RACE RESULTS / Jim Bohle 39
Scoring a major marathon with dBASE II

UTILITIES: POLYLIBRARIAN / Object Module Maintenance 42

SOFTWARE REVIEWS
dWINDOW / Jim Bohle 43

PJ LIBRARY 45

MISC. 46

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOUSEKEEPING

HOUSEKEEPING

How do you feel when you have finished reading a computer magazine? Are you ready to buy a new product? Have you learned something? A magazine comes in the mail and you spend an hour or two with it. The emotions that you experience after setting the magazine down will tell you a lot about the content of the publication. Have you picked up some valuable information? Did you met other programmers like yourself? We spend too many hours alone on the keyboard. When it is time to come up for air, we hope that the encounter gives us the shot in the arm we need.

1984 is the Year of the Programmer. Go for the gusto. It is high time the United States Postal Service acknowledged our contributions to the computer revolution. Let's have a computer generated graphics stamp to promote the celebration.

There is a "brain drain" happening in the computer book market. Book publishers will seek out the most knowledgeable computer people and insist that they write novice books. I support the need for experienced people writing novice tutorial texts but not all the time. Many universities insist that every full professor teach one beginning class each quarter. It keeps them in touch with young minds, and gives entering freshmen the opportunity to experience the enthusiasm that professors have for their chosen discipline. The professors spend MOST of their time teaching graduate students.

If you call up a computer book publisher and ask them about doing a manual on bit manipulation in the LISP language they are likely to turn you down. They will immediately suggest you do a "Beginners guide to the Monochrome Contrast Knob." I am overstating the issue. We need more intermediate and advanced books for the IBM PC. Sure, write a novice manual to pay the bills, then do a book that uses your abilities to the fullest. OK?. . . .

The difference between "involvement" and "commitment" to a software project is a deciding factor when a deadline is approaching. Working in a group you soon learn who is involved and who is ultimately committed. The same concept applies to business or personal relationships. If the distinction is a little vague, an analogy may help. When you think of involvement and commitment, think of a "ham and eggs breakfast." The chicken is involved, the pig is committed.

— Greg

PS. We have had requests for articles on a number of technical topics. If you are writing code that covers any of them, let us know. Here are a few that have been mentioned: Sort routines in C, chaining among C programs, BASIC routines for checking valid dates or providing the day of the week, effective use of TABLES, personal machine & software setups to maximize development time, etc.

PROGRAMMER'S JOURNAL

Editor/Publisher: Greg Estes
Associate Editor: Warren Hillman
Contributing Editor: Jim Bohle
Art Director: Julia King
Production: Cassandra Estes
Copy Editing: Renee Anderson

Contributing Writers:
Morton F. Kaplon
Robert Dunaway
Ed Keating
Ken Smolek
Brett Salter
Rick Millhollin
Dan Rollins
Paul Barkley

Editorial and Business Office:
2765 Potter St.
Eugene, OR 97405

Mailing Address:
P.O. Box 5070, Eugene, OR 97405

Programmer's Journal. Volume 1, Number 6.
Copyright (C) 1983 by Programmer's Journal
Published bi-monthly.

Postmaster send address changes to:
Programmer's Journal
2765 Potter St.
Eugene, OR 97405
ph. (503) 687-0398

Subscriptions: Bi-monthly rates are $24 U.S.
($30 in Canada, and $40 elsewhere.)

IBM and Personal Computer are trademarks of
International Business Machines, Armonk, New York
dBASE II is a trademark of Ashton-Tate.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

MULTI-JOB SOFTWARE DEVELOPMENT

I read with interest the multi-tasking review in PJ #5 because I had just installed B&L's Multi-Job. To Frankie Detesto's comments, may I add that Multi-Job is an exceptional productivity tool for software development. It is also reliable, and hasn't crashed on me yet. I did run across a couple of quirks, discussed below. Readers who have doubts about the effectiveness of multi-tasking on these little computers, as I did, may be interested in my experience.

I am running a 640K XT which has been configured as follows:

AST Spooler 32K
AST Memory Disk 160K
Multi-Job 1 195K
Multi-Job 2 115K
Btrieve buffers 32K

Notice this does not add up to 640K. There is some Multi-Job overhead for each task, but from 60 to 80K vanishes over and above what the documentation covers. B&L is researching this problem.

The other quirk is that Multi-Job intercepts extended key codes when DOS interrupt 21h is used. This means that function keys are delayed by one keystroke until a non-extended key is pressed, at which time the currently delayed key as well as the key just pressed become available. It is like the echo we hear occasionally on some telephone networks, and just as disconcerting. Bob Hudson, the program author, responded to my letter with a phone call and a work around it suggestion — use the interrupt call 16h. He acknowledged that this is a bug in Multi-Job to be fixed, I gather, in the future. Fooling around with this bug, I found that only my software was affected, commercially available products like Personal Editor and VisiWord worked without a hitch. Oh well, add another to the "Real Programmer" list. R.P.'s don't USE interrupt 21h!

Multi-Job is so productive in my environment because it allows me to toggle instantly between Pascal compilations or assemblies and my Personal Editor. If a job bogs down, which will happen with 2 compute-bound jobs going against each other, a single keystroke quiets all the other partitions, and sends the foreground into hyper-drive. Another keystroke reactivates multi-tasking. This control adds greatly to Mutti-Job's productivity. When I need speed, I lock out the background. While I am pondering the screen, a compilation is bumping along in the background.

B&L have done a good job with this product, and they respond quickly to questions. I consider this tool essential to my work.

I would like to add that solely on the strength of the reviews of editors in a prior PJ, I purchased the Personal Editor. I was not disappointed, but DID have to drive all over town looking for a store which wasn't out of stock.

Finally, I would like to add my strong recommendation for SofTool Systems BDS product (reviewed in PJ #5), which I have used for over a year. If you do anything in BASIC, you'll wonder how you ever got anywhere without BDS and still maintained your sanity.

Sincerely,

Chet Floyd
F/S Associates, Inc.
Manhattan Beach, CA

MULTI-JOB, B&L Computer Consultants, 7337 Northview, Suite B Boise, ID 83704 ph. 208-377-8088

C SCREEN EDITOR

CSE: A full-screen text editor written in C

$60 Includes the complete source code

- Powerful command set includes cursor control, find/replace, block move, file inclusion, and nested macro commands.
- Full-screen editing, including cursor control, insert/overtype, single and multiple line insertion and deletion, and forward and reverse scrolling.
- Commands to perform large-scale operations on the text buffer, such as find, find and replace, move the text pointer, read input, and write output.
- A macro facility which allows grouping of a set of commands for repeated execution.
- Ability to work on files larger than the text buffer size.
- Complete configurability of control sequences, such as cursor keys and editor control strings, to allow customization of CSE for a specific terminal. CSE is configured to use special editing and function keys on the IBM PC.

NMD Northwest Microsystem Design

P.O. Box 10853 • Eugene, OR 97401 • (503) 484-7129

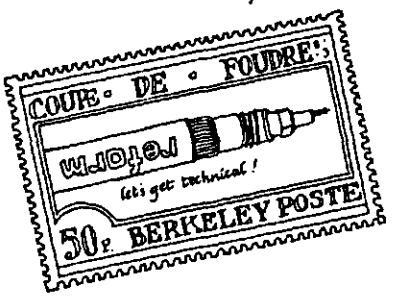
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ GRAPHICS

Enclosed is a check for $48. Please start my subscription with the first issue.

I don't have a PC (IBM that is) but I was so taken with the extraordinary graphics and the excellent articles that I'm subscribing anyway. There's no reason that technical magazines can't be beautiful too — and you're proving it.

Tom Meadow
Berkeley, CA



PC - APPLE CONNECTION

Thank you for sending me a complimentary copy of your publication. Enclosed you will find my check for a new subscription. Perhaps you could help refer me to one of your advertisers or some other source — possibly an article which has appeared or will appear in the Journal — to solve some problems I am facing.

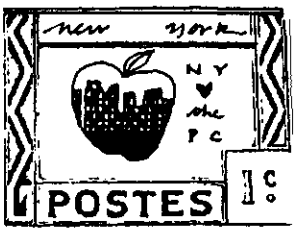
By way of background — I have extensive experience in data base programming on the Apple in Applesoft, rather less capable in assembly or Pascal. I now am trying to convert (or re-write) a medical office billing system I wrote for the Apple to the IBM PC. I have several problems, some serious, some trivial.

First, there does not seem to be any way for a Basic program to clear to the end of line or to the end of screen! I certainly can calculate the number of spaces to the end of the line or the number of lines to the bottom of the screen, and PRINT the correct number of spaces, but that is slow and causes the screen to blink. There must be a better way (my Apple just issues calls to -868 and -958 in the monitor). Nor is there an equivalent to Apple's ability to set the text window to protect the first 3 rows, rightmost 6 columns, etc., by way of POKE 34,X etc. No way to scroll part of the screen only. Am I the only one who needs routines like this?

Garbage collection is not a problem on the Apple if you use Wigginton's Fast Garbage Collections (CALL-A.P.P.L.E., Jan, 1981) or Bongers' Memory Management (Micro, Aug, 1982). Is there any equivalent for BASICA on the IBM? Would a different BASIC be better?

The above issues are serious, and affect how or whether I go ahead with my project at all. Perhaps I should consider Pascal, or even Modula-2, instead of Basic, although I prefer the easy modification of Basic. If you can share with me whatever thoughts you may have about these two latter languages and their implementations, I certainly would appreciate it. In addition, there are several file management subroutine packages being advertised (ISAM, BTRIEVE, BTREE+, FABS); do you have any opinions about these, or do you (will you) plan a comparative review? This is less critical, since I have a fairly adequate set of routines I have developed myself on the Apple; I would hope, however, that I could do better on the IBM. Likewise, on the Apple, I use a very useful basic line editor called GPLE, which edits lines, does global find, search, and replaces on a program (either on variables or on strings, etc.), has definable macros, etc. What is available on the IBM to make programming chores easier?

Jerome B. Blumenthal, M.D.
Binghamton, NY



Jerome: Any time you use POKES, you are programming at a very untransportable level. The same goes for using screen handling routines in a direct BIOS fashion. If you wish to once and for all remove yourself from the situation you are in then you must develop a set of BASIC routines to solve those problems. Then in a year or two when you want to transport your Medical system to a NEW computer it will be simpler. Go ahead and count the spaces you need cleared and erase text that way. If speed is what you want I would try the BASIC Compiler from IBM.

The BIOS and DOS lower level service routines are much more extensive than the Apple. You can do windowing, clear to end of screen, etc. Several IBM PC books discuss them. Try Peter Norton's manual on "Inside the IBM PC."

For CLEARing excess garbage buildup in a BASIC program you can use the CLEAR or ERASE command. The CLEAR command sets everything back to zero or null, all arrays and variables. Use it if you want to restart the whole program or begin a new cycle. The ERASE command will allow you to erase arrays and reallocate them.

The problems you have mentioned are the usual ones that should be solved within the language you currently use rather than changing languages. Each language has its good and bad points, you gain something and lose something whenever you switch. Have you tried using a data base manager like dBASE II? If you do normal batch style processing of your records it will save you weeks of programming. You could use such a data base system for 90% of your processing and when you want a

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

special report just convert the data base file to a standard text file and run your BASIC programs on it. Using ISAM and BTRIEVE utilities will certainly work but you are once again going outside of your language to solve your programming needs.

There are a world of good editors on the IBM PC, see our issue #3 for a review of several of them. — ED

INTERRUPTS

I enjoy your journal. Just keep watching for the Yellow Zone interrupts (DEC PDP-11 nonsense for stack errors).

J. Edward Volkstorf, Jr.
New Venture Systems
Chesapeake, VA

THE ULTIMATE TRUTH

"Real Programmers can write FORTRAN in any language"

Ed Keating
Arlington Heights, IL

CHECK THIS OUT

Deer PJ,

Eye donut know watt two tell yew. Dish soft wear spilling checkers no mower good four me. Know matter witch weigh i right ore spell wierds it due knot fix dam, knot at all. Wood you warn utter users to bee berry care full a bout gnu soft wear. Luke watt eat deed two me.

Eyesic Askimof
Sri Lanka

PROKEY CAUTION

Trying to define certain keys using Prokey will cause you headaches. Here are the ones to watch out for.

Ctrl-H is the symbol for the backspace.
Ctrl-I translates into a TAB.
Ctrl-M is the same thing as the RETURN key.

PC POST
Modesto-Turlock IBM PC Users Group
Modesto, CA

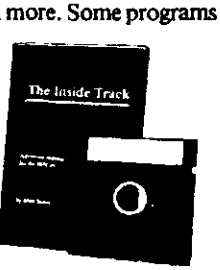
□

Know Thy PC!

Are you writing programs in BASIC or Pascal? The popular **Peeks 'n Pokes** has a disk with 58 programs and a 38-page manual that helps you get 'underneath the covers' of the PC. Learn how to use PEEK, POKE, INP, OUT, and DOS/BIOS function calls to do what you want, fast! Do you want to perform functions not available from BASIC or Pascal? It's all explained in the manual and demonstrated in the sample programs. Source code included!

Peeks 'n Pokes shows you how to:
- Access the system's configuration
- Unprotect BASIC programs
- Scroll part or all of the screen
- Access the file directory
- Logically swap printers
- Read and change the keyboard
- Find more Peeks and Pokes
- And much more... for only $30.00

Want to know more? **The Inside Track!** is a collection of advanced utilities for the PC programmer. It contains a disk with 61 programs, a 42-page manual, and a fold-out memory map that helps you get better performance from the PC. With this package you can give your programs assembler-assisted speed from high-level languages, get control over memory, customize and control the PC, and more. Some programs require DOS 2.00. Source code included!



The Inside Track! shows you how to:
- Read/write files as fast as DOS
- Display data on the screen faster
- Reserve memory for your use
- Copy memory to another location
- Copy-protect your programs
- Load large programs faster
- Control the keyboard settings
- And much more... for only $45.00

MasterCard and VISA accepted. Shipping charges: $2.50 per order for UPS; $2.50 per item for First Class Mail to USA and Canada; $6.00 per item for Air Mail outside USA and Canada. Dealer inquiries invited.

Data Base Decisions • 14 Bonnie Lane • Atlanta, GA 30328 • 404/256-3860

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ PROFILE

BOOKS AND SOFTWARE

An interview with Dan Rollins, author of a new book on IBM PC Assembly Language Programming and numerous magazine articles for BYTE, PC, PCA, and PJ.

PJ: When did you get your first computer?

DAN: My first one, years ago, was a TRS-80. I had been looking at computer for a while and then the price dropped down to $499. I thought that if I could ever afford one, I could afford it now. It had 4K of memory. The decision rested with my ability to think up 10 uses for the thing. If I could come up with at least 10 legitimate reasons then I would go ahead and buy one. Pretty soon I was saying to myself, if I could only think of 5 good uses I would go ahead and buy one. That was about 6 years ago.

PJ: How long have you been doing free-lance articles?

DAN: It seems like a very long time, but now that I think about it, I've been doing it full time for about a year. Mainly my work has always been writing, usually describing a program. You know, here is a program, this is what it does. I sold my first articles to MICROCOMPUTING magazine. Eventually I sold an article to BYTE magazine and realized that it was possible to do it full time. BYTE convinced me I had a good chance of succeeding.

PJ: When did you buy your IBM PC?

DAN: Let's see, in July of last year. I bought one of the first ones that had double-sided drives.

PJ: What languages have you learned?

DAN: BASIC was my first language on the TRS-80 and then I learned assembly. I was very turned off by it at first. The tools I had for programming in assembly were very primitive. You would load in one program from the cassette tape which in turn would load in another program that was the assembler. If anything went wrong you had to go back to zero and try it all again. Then you would save the assembled code onto tape, reload the debugger, reload the assembled code and run the tiny program that made your screen go white. It was very frustrating.

On the IBM PC the Macro Assembler was the first language I bought. I was also amazed at the quality of the DEBUG utility.

PJ: Have you seen TRACE86 from Morgan Computing?

DAN: Yes, as a matter of fact, I did a review of it for Personal Computer Age. It has more going for it than DEBUG. It is written especially for a full screen display rather than a teletype terminal like DEBUG. TRACE86 displays the registers all the time, you watch them changing and the stack increasing and decreasing. You press the space bar to single step through the ASM code.

PJ: TRACE86 seems to give you an impression of program flow that is not as apparent when using DEBUG. TRACE86 is immediately educational, like the Norton Utilities, as well as being a professional tool.

DAN: Absolutely. It is a great tutorial. If you load TRACE86 and press nothing it goes into a demo mode that shows you how it works. It sells itself. The demo goes through a loop and you can SEE a loop happening. This eliminates the need for any additional explanation. The dynamic display of the stack is good too. As you PUSH something onto it, it gets longer and longer. You see what a stack does.

PJ: How hard was it to learn 8088 assembly language?

DAN: With the Z-80 behind me it was not too bad. Certain things were the same. The registers correspond. I did have to get used to the increased flexibility, like incrementing a memory address. I did a lot of dis-assembly to start out with. The ones that come with the DOS diskette, EDLIN and others. I wrote a fast screen display routine that interfaced with BASIC for the July issue of BYTE and another one that illustrated the USER function.

PJ: When you dis-assemble a piece of code, isn't it important to be aware of whether it was originally written in assembly language or created from a compiler?

DAN: Definitely. EDLIN, for example, is written in assembly. The advantage is that programs of this type are written by professional programmers. You see modular subroutines. A subroutine is called and you can figure out where it is going, what parameters are passed, etc. It is very educational. Dis-assembling compiled programs is something like confronting a large maze of code. Finding you way around is a nightmare.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ: Who pays the best for magazine articles?

DAN: The most money I received was from BYTE. It was an unusually long article with a lot of code. The article was on shape table graphics for the TRS-80. The payment was around $1400. This fee was large because the article was quite long with 4 enormous listings. BYTE makes payment based upon the published page. Now BYTE has started condensing listings to reduce the page count.

Most of the other publications make a general payment of around $200-$500. Some publications pay by the word, 10 cents a word. So a 2,000 word article will bring in $200. Reviews are often shorter and may not pay as much, but you get to keep the software as part of the project.

PJ: Who is publishing your assembler book?

DAN: Macmillan & Co. They have a growing line of computer books.

PJ: A growing "brain drain" seems to be occurring in the computer book field. Book companies will take knowledgeable programmers and ask them to write novice books. Obviously the sales figures for a novice book will be higher then those of an advanced manual. Must the advanced topics always suffer to the book companies/writers desire for higher profits?

DAN: I have been steered repeatedly away from doing advanced books by publishers. They want the novice DOS style manuals. These are not the books I would like to write. Hopefully the publishers could up the price on a technical book and still garner the profits they want.

PJ: What are the royalty rates like?

DAN: They are on a sliding scale. A certain amount for the first 10,000 books (say 6%) and then a higher rate for the next several thousand. They have a number of different scales. Eventually you should arrive at 10%.

Shopping around will give you an idea of what to expect in a contract. Be up front about the fact that you are not committed yet. One company is mad at me because of this. I was trying to find the best contract and they were unaware that I was still looking around.

PJ: What are the contracts for short articles like?

DAN: I used to get a single sheet of paper in the mail with no carbon copy that said something along the lines of, yes we would want your article, for $40, check the box (yes or no) and mail it back. Contracts you get now are much more explicit and mention first world serial rights. Serial rights refer to magazine publication.

There is usually a second paragraph that covers anthology rights or reprint rights, in case they plan on combining your article with others to form a book. Generally they say ALL rights in contracts. This means both serial and anthology rights. Sometimes I will cross out anthology and see if they will still pay me the same as they originally agreed to.

PJ: Have you done any graphics assembly work on the IBM PC?

DAN: I have the color card and have done a considerable amount of assembly graphics. There are several chapters in my book about it. One is called Text and Graphics, but most of it is on graphics. Once you explain how to draw a character on the screen, it is the same for both monitors. The graphic discussions involve shape, line and circle drawing. There is a game out now for the IBM called Sea Dragon (from Adventure International). I did the conversion (a COMPLETE REWRITE!) for the PC.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Conversion programming is something that many people are doing. The royalties are not as high as writing the game yourself, but you sell more games because of the previous name recognition. Some games merely require converting PRINT @'s to LOCATE's. The more assembly language involved the harder it is. Sea Dragon was all in assembler, no higher level language at all. The translation got extremely hard because there was no documentation with the program. You can tell what the game does by playing it, but I am not a good game player. I could never get to the end of the game to find out what the last scenes were supposed to look like! I finally had to modify the game so I got 255 ships to fight with rather than the usual 3. It gave me a bigger edge in finding out what was going on. Constructing a program logic chart helped a lot too.

PJ: There are a number of phrases used to describe lower level functions on the IBM PC like service routines, function calls, interrupts, DOS calls, and ROM-BIOS routines. They have been used in an overlapping fashion which has lead to some confusion. Could you shed some light on the issue?

DAN: "Interrupt" is a broad word. It has a number of uses, all valid. Just like the word Segment. You can have a 64K block of memory described as a Segment, or something can be called a "logical" Segment, or it can be a Segment Register.

An interrupt is a short hand way to make a call. You want to CALL routine number 3 let's say. You do not know where routine number 3 is in memory. You must indicate in one register which routine you want, in this case 3. Then you often have to load other registers with parameters you are passing.

There are also hardware interrupts. A piece of hardware can interrupt the 8088 and cause it to stop what it is doing and go do something else.

The ROM-BIOS interrupts are the very lowest level ones. They put characters on the screen or read the disk.

PJ: The ROM-BIOS of course shouldn't be used if you want your software to be portable.

DAN: Right. You wish to stay away from them unless you want to do something special for a particular machine.

The next level up from this is interrupt 21H or the DOS services. You can do more sophisticated things with these routines. Often they just echo the ROM-BIOS lower level functions. The purpose here is to add a layer above the BIOS so that you are immediately removed from the hardware level contact.

PJ: How about service routines and function calls?

DAN: Those almost always refer to what you are accessing when you are using interrupt 21H. You request whichever routine or function you want by loading register AH with the appropriate number.

PJ: Do you balance programming and book writing on a weekly basis?

DAN: A book is much more critical than an article. For long texts I set up a weekly or monthly goal to mark where I should be on a certain date. I try and put in an 8 hour day. If I do not finish as much as I wanted one day then I have to put in more hours the next. If the weekend comes and I am far behind then I will put in 12 hours if I have too. It is not a big strain for me to sit in front of the screen for 12 hours at a time. I am accustomed to it by now.

PJ: Which monitor do you use?

DAN: I am using an inexpensive green monitor with a color card. Everybody tells me not to work with the monochrome unless I have the money to buy it. Once you use it you are hooked.

PJ: How long is your ASM book?

DAN: About 250 pages. It took me about 6 months to write it. The publisher's contract has a fixed date of completion. After that time you are in real trouble. I am not sure what they do but I would rather not find out.

I am not doing much conversion programming these days, mostly writing. Often I will come up with a software idea that may not evolve into a complete program, but is an ideal section of a larger program. These kind of ideas work well for magazine articles.

PJ: Do you plan to buy a PC-Junior?

DAN: I am signed up to receive one of the first ones that arrive here at the local Computerland.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Another machine I am interested in is the HP-150, the one with the touch screen. It should sell a lot to the business community. Computer novices will be drawn to it.

PJ: Do like writing more than programming?

DAN: I like both. Each of them involves putting symbols together in a particular order. One is much like the other. It is nice to be able to combine the two. □

INTRODUCING

POLYLIBRARIAN™

THE OBJECT MODULE LIBRARIAN

FOR PROGRAMMERS USING MS-DOS (PC-DOS)

POLYLIBRARIAN lets you create a single library from related Object Modules. The IBM PC Linker will then select only the modules necessary to produce an executable file.

For added flexibility, POLYLIBRARIAN works with any compiler or assembler that uses the MS-DOS (IBM PC) Linker.

POLYLIBRARIAN lets you use your time more productively because you'll eliminate repetitious programming steps.

- Shrink Development Time
- Optimize Object Modules
- Unlock The Secrets of Existing Libraries

Three Operating Modes

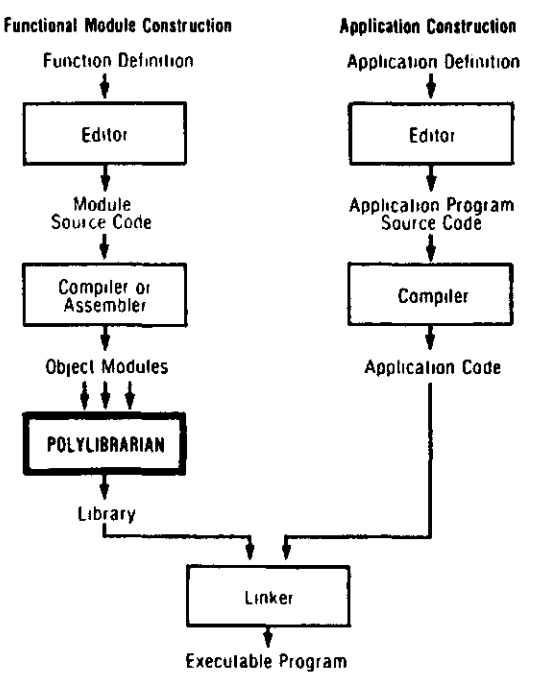
The Interactive Mode, Command Line Mode, and Command File Mode allow first-time librarian users and advanced users to realize the full power and potential of POLYLIBRARIAN. The Fully Nested Command Modes have built-in help features.

Comprehensive User's Manual

The highly praised User's Manual explains the functions of a librarian, describes the use of each operating mode, includes a Command Summary, list of error messages, the Intel Object Module Format, and a complete BNF description of the POLYLIBRARIAN Grammer. A step-by-step tutorial lets you learn to create libraries, add and delete modules, extract modules and more.

Requires 128K RAM
MS-DOS 1.0, 1.1 or 2.0

The Missing Link in Structured Programming



VISA/MC ORDERS CALL
1-800-547-4000
ask for Dept. 306

$99
(VISA ADD $1.50)

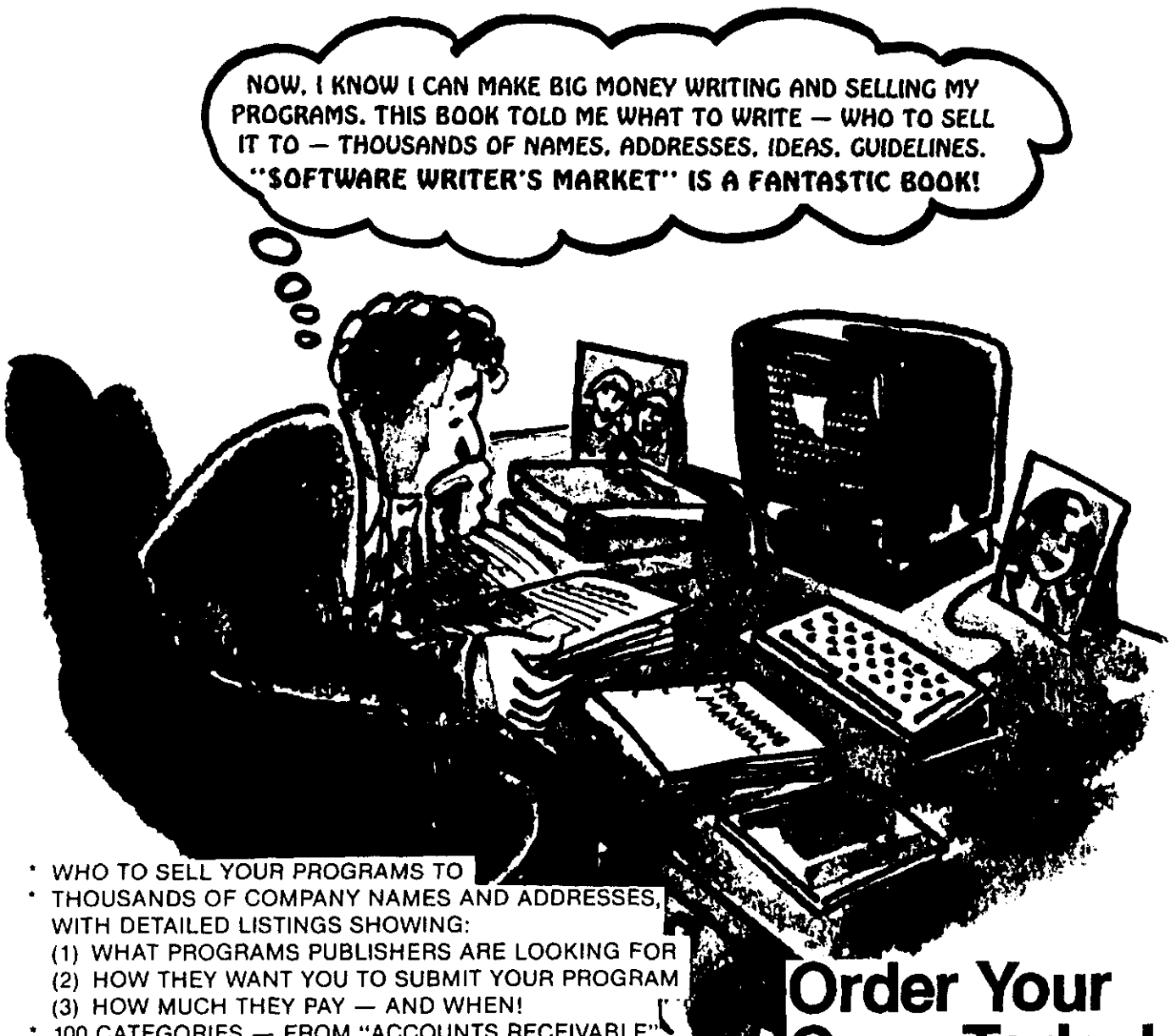
POLYTRON™

OR SEND CHECK, P.O.'s TO: DS-3-306 P.O. Box 787 Hillsboro, OR 97123 (503) 648-8595

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer s

READ THIS...


NOW, I KNOW I CAN MAKE BIG MONEY WRITING AND SELLING MY PROGRAMS. THIS BOOK TOLD ME WHAT TO WRITE — WHO TO SELL IT TO — THOUSANDS OF NAMES, ADDRESSES, IDEAS, GUIDELINES. "SOFTWARE WRITER'S MARKET" IS A FANTASTIC BOOK!

- WHO TO SELL YOUR PROGRAMS TO
- THOUSANDS OF COMPANY NAMES AND ADDRESSES, WITH DETAILED LISTINGS SHOWING:
  (1) WHAT PROGRAMS PUBLISHERS ARE LOOKING FOR
  (2) HOW THEY WANT YOU TO SUBMIT YOUR PROGRAM
  (3) HOW MUCH THEY PAY — AND WHEN!
- 100 CATEGORIES — FROM "ACCOUNTS RECEIVABLE" TO "GAMES" TO "VIDEO CONTROL" PROGRAMS
- HOW TO WRITE CLEAR DOCUMENTATION
- DEBUGGING TECHNIQUES

Order Your Copy Today!

Enclose check or money order for $19.95 (No C.O.D.'s) to:

IPF Publications
146 Country Club Lane
Pomona, NY 10970
(914) 354-5585

Name .....

Address .....

City..... State..... Zip.....

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

STRUCTURED PROGRAMMING REVISITED

by Paul A. Barkley, consultant, author, and programmer

I was working on a chapter about structured programming, in a book I'm doing about tips and tools for IBM PC programmers and professional consultants when Volume 1, Number 4 of the "Programmer's Journal" arrived. Joe Rayhawk did a good job on structured programming, but after reading Part 2 of his article in the following issue, I felt that there were some items that warranted further discussion. Having taught college classes as well as making a living in the real world of IBM PC programming and consulting, I believe we need a bridge between academic theories and techniques that really work.

DO-WHILE VERSUS DO-UNTIL

The furor in the academic world over the use of DO-WHILE versus DO-UNTIL is just that: academic debate. In teaching BASIC programming at a local community college, we try our best to force students to adopt a structured, logical approach to program design. Part of the process is, of course, to show them the three main structures: SEQUENTIAL, IF-THEN-ELSE, and looping. While the concept of looping is straightforward, it is obviously important to show the students how the procedure ACTUALLY works in the particular language and dialect that we are teaching (unfortunately it's BASIC implemented on a large and motley collection of Commodore PET's, and a few mangy Trash 80's).

The students follow the discussion real well as I sketch the DO-WHILE flowchart. They observe the flow starting with the decision box which branches on TRUE to the interior of the loop and branches on FALSE to the next structure in the program. It makes semantic sense to say the loop executes WHILE THE CONDITION IS TRUE. In other words, we have a DO-WHILE-TRUE structure.

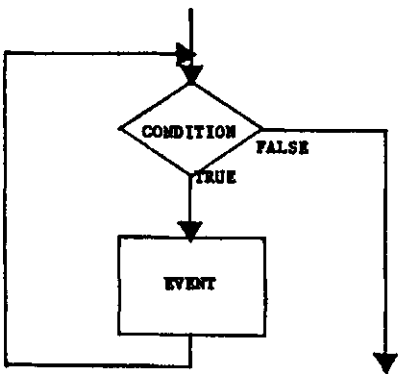


Figure 1: DO-WHILE

In our courses, we don't introduce the concept of structure until about half way through the term. The students have already written a bunch of programs utilizing an inevitable spaghetti approach and have spent hours in the lab trying to get the damn things to run. Right or wrong, and many institutions disagree with this approach, we offer structured programming as a solution to the "natural" method of programming. Therefore, the students already know how a FOR-NEXT loop REALLY works and have been burned by having the loop execute once even when this was not desirable.

A simple example is to write a program to read twenty five quiz scores between 0 and 20 and print out a horizontal bar graph made of asterisks, with one asterisk for each point scored. The easy way to do this is to use a loop with a variable — the quiz score itself — as the termination point. The content of the loop is to simply print one asterisk with a hanging semi-colon to prevent a carriage return. This illustrates nicely the concept of nested loops and variable controlled loops:

```
100 FOR K = 1 TO 25
110     READ QS
120     FOR J = 1 to QS
130         PRINT "*";
140     NEXT J
150     PRINT
160 NEXT K
170 DATA 15,7,0,1,20,19....
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The problem, of course, is that some bozo is bound to get a zero on the quiz, and their bar chart will still be one asterisk long, the same as a student who actually scored a one on the quiz. The student attempting to solve this problem will therefore have to insert an IF statement to check for QS = 0 before entering the loop (it's bad news to keep jumping out of loops in the middle, and Commodores don't tolerate much of this.)

```
100 FOR K = 1 TO 25
110     READ QS
115     IF QS = 0 THEN GOTO 150
120     FOR J = 1 to QS
130        PRINT "*";
140     NEXT J
150     PRINT
160 NEXT K
170 DATA 15,7,0,1,20,19....
```

Consequently, while I'm scribbling the DO-WHILE structure on the board, and taking pains to mark the TRUE and FALSE exits properly, usually one of the brighter students will point out that this doesn't seem to be the way that the FOR-NEXT loop really works. At this point they get an 'Attaboy" (occasionally an "Attagirl") and the rest of the class starts going glassy eyed.

In the flash of an eye the chalk scribbles out the DO-UNTIL structure with the contents of the loop coming first, FOLLOWED by the decision box. Regardless of how many times I've done this, I always get this little sinking feeling as I mull over the semantic implications of the structure and attempt to label the TRUE and FALSE paths, while the class waits with baited breath (whatever that cliche means — does a lizard with a sticky tongue hanging out wait with baited breath?). Semantically, the loop should continue to execute UNTIL THE CONDITION IS TRUE. Joe Rayhawk properly stated this in his article, but the flowchart was mistakenly labeled backwards.

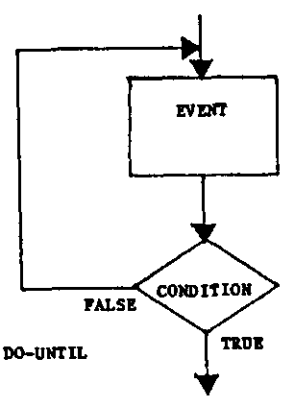


Figure 2: DO-UNTIL

What does this suggest to us? That Joe is wrong? That I should use note cards until I learn to remember which path is which without pondering the philosophical implications? What it suggests, of course, is that all this is somewhat arbitrary. In fact, in writing the code for a given comparison, it's often easier to check for an opposite condition than to check for the condition that we're logically interested in. This means, of course, that if we were to flowchart the situation, the TRUE and FALSE paths would be backwards, and in fact this sort of arbitrariness no doubt led to the mis-labeling on Joe's flowchart.

Joe points out an important caveat, though, and that is that the PC doesn't implement the FOR-NEXT loop like most machines (i.e. as a DO-UNTIL). I personally find this a bizarre departure from most BASICs, and since the PC already has a WHILE-WEND structure for DO-WHILE programming, I really can't see any reason for it.

Now for the big question: does it really make any difference WHICH structure a particular BASIC supports? Not really! As long as the programmer is aware of EXACTLY how a loop is executed in the dialect in which he or she is working, there are usually as many times that one structure is slightly more useful than the other, than the reverse case. In other words, who cares besides academicians? Having always used the traditional FOR-NEXT loop, I personally never bother with the WHILE-WEND, although clearly it's every bit as useful as the traditional FOR-NEXT. Anyone who spends a great deal of thought choosing one structure over the other every time a loop is called for, probably isn't getting a whole lot of coding done.

NEW FANGLED LOOPS

I was fascinated by the GENERALIZED LOOP and the ERROR CHECKING LOOP that Joe described, because I had never seen them before. I'm not an academician, as you might have surmised from some of my other comments, so I always suspect that there are a lot of things that exist in Ivory Towers that the rest of us don't know about. My first thought was that Joe knows something I don't, and my second thought was that Joe just made up these two structures to bug everybody. Either way, they illustrate a fascinating, and often lethal problem with structured programming: 'pure" structured programming is a pristine discipline that will force you to drop some of your ill-mannered, and probably self-taught bad habits, but it can be a ludicrous pain in the ass.

I don't have much use for zealots of any sort, from the Spanish Inquisition to the Ayatollah Khomeni, so dyed-in-the-wool structured programming afficianados leave

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

me cold. I've had too many lunches ruined by fellow instructors who propose exiling anyone caught using the infamous GOTO statement to Devil's Island, Siberia, or New Jersey (you pick).

Well, theory has it that ANY program can be written using just the basic three structures: SEQUENTIAL, IF-THEN-ELSE, and DO-WHILE (or DO-UNTIL). The theory also says that if you DON'T use these structures, you'll surely end up in programmer's hell. Obviously, then, in a complex program the contents of a loop, or the contents of each of the two branches of the IF-THEN-ELSE can be a whole structure in themselves. If you play around with this, you'll find that anything you can think of to program can, in fact, be set up this way, including the so-called GENERALIZED LOOP and the ERROR-CHECKING LOOP. The problem is that in certain cases, it takes a lot more statements to do it. Thus, preserving the concept of structured programming may not be worth it.

Certain programming tasks are performed often enough, that they have warranted the creation of their own special structures. A good example is the CASE structure, which makes much more sense than using a whole bunch of cascaded IF-THEN-ELSE statements. It's easy to see, easy to understand, and preserves nicely the idea of a structure having one entrance and one exit. It's an example of some structured programming enthusiast recognizing reality and putting the seal of approval on a new structure with the same spirit as the original structures.

The same argument can be applied to the GENERALIZED LOOP and the ERROR-CHECKING LOOP, but quite frankly, they look like an attempt to legitimize a relatively frequently used code implementation that just won't fit into our nice pattern of structured programming. Since it won't fit, an orthodox priest of the structured programming cult (SPC) will call for its abolition, while a reformed SPC priest will cleverly add it to the collection of structures, thereby defusing criticism from the always pragmatic lay fold. If you add enough "structures" to the collection, the concept of structured programming will become meaningless. I suggest that while a good programmer will embrace the concepts of structured programming, no one but a quiche-eater would swallow the entire dogma. In other words, use it as much as possible, then go ahead and solve the problem your own way when it no longer serves its purpose.

SUBROUTINES AND STRUCTURED DESIGN

Joe lists the SUBROUTINE last in his article, and calls it a structure EXTENSION. OK, maybe that's what the SPC priests call it, but it's probably the most useful of all these "structures".

Those who invent all these interesting computer science ideas, made a bad mistake when they invented STRUCTURED PROGRAMMING and STRUCTURED DESIGN. The names sound similar, and leave students with nothing to easily distinguish them. Whenever the class curve gets too high in an intro class, a couple of questions freely mixing the terms is guaranteed to lower everyone's score.

Structured DESIGN is the process of organizing a program into logical discrete modules which can be presumably written and debugged in a certain amount of isolation. I work on programs where I'm the system analyst and 100% of the programming staff, and I shudder when I think about a team of systems analysts and 50 programmers, all with good sized egos, starting to work on a large program. Everybody has to do their part, by themselves, and then all the pieces have to be

TELECOMMUNICATIONS TYPESETTING

COMPUTER OWNERS: If you have a modem, text editor, and communications program, you should consider the improved readability and cost savings provided by *TYPESETTING* your program documentation, manuscript, newsletter, or other text instead of just reproducing it from line printer or daisy-wheel output. Computer typesetting offers you high quality, space-saving copy that creates the professional image you want! Hundreds of type styles to choose from with 8 styles and 12 sizes "on line." And it's easy to encode your copy with our designated typesetting commands.

COMPLETE CONFIDENTIALITY GUARANTEED
— bonded for your protection —

PUBLICATION DESIGN, EDITING, & PRODUCTION

Editing & Design Services Inc.

30 East 13th Avenue Eugene, Oregon 97401

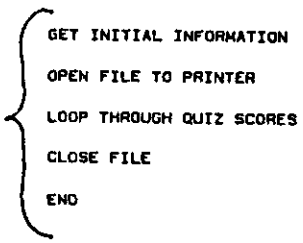
Call Collect 503/683-2657

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

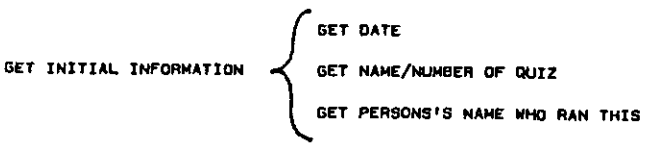
integrated, and of course they never quite fit together properly. Nevertheless, without a structured design of some sort, you couldn't even begin such a project.

In our Academic (what a bad word) Data Processing department, we teach a thoroughly despicable technique called the Warnier (WARN nee ay). This obnoxious technique, which no one I know uses, but which we force on our students, is only ugly and useless because it gets out of hand faster than a flowchart. The basic idea is absolutely precious.

Warnier diagrams work like this. You start with a column at the left. You begin writing down the steps that will be performed in the program, using words that express the general IDEA, instead of many lines of code that actually accomplish the task. For example, the steps to be performed in the quiz bar chart problem discussed above, would be:



This is certainly not enough information to begin coding the problem. Each of these steps must be expanded to better illustrate what we mean. In order to break each piece up, we move one column to the right and repeat the process. For example, expanding upon the initial entry:



As we can see, we still aren't to the level of coding, but the next column will be trivial as we break each piece up. Some people quit here, but the idea is to carry each entry to the right, gathering more detail until we have ALL the code we need. For example, GET DATE would be expanded to something like:

```
INPUT "What is today's date (i.e. 10/15/83)" ; D$
```

Generally, students aren't very "user-friendly" , so we usually see lines like:

```
INPUT D$
```

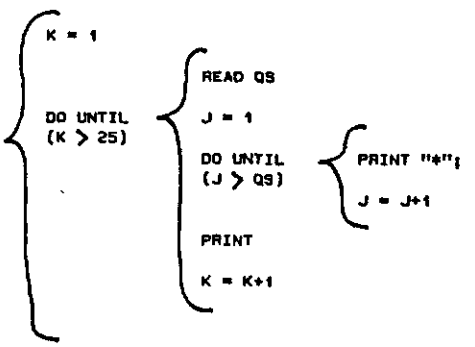
Of course, having gone to the trouble to do the Warnier, most people can supply the "polish" of meaningful prompts.

Note that in expanding OPEN FILE TO PRINTER, we can move immediately to actually coding the line, with something like:

```
OPEN "LPT1:" FOR OUTPUT AS #1
```

This illustrates the fact that the actual coding winds up in all sorts of columns, and that three or four or five is not a magic number for the amount of detail necessary. Naturally, larger programs are likely to have more columns in order to explain what is needed.

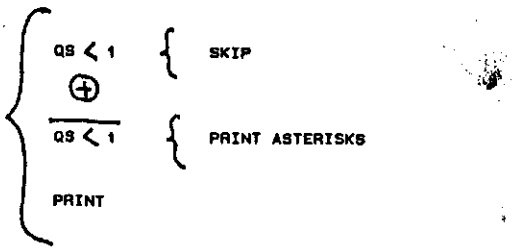
This is not a Warnier tutorial, so we don't want to spend much more time on this, but note that all we have illustrated so far is the SEQUENTIAL structure, with ideas going from top to bottom down a column. All of the ideas in the next column to the right that explain a single item must be accomplished before the next item can be done. Warniers also use the brace to enclose an entire loop which is labeled as a DO-UNTIL or a DO-WHILE on the outside of the brace with the appropriate condition stated, or they use explicit initialization, incrementation, and comparison of variables to accomplish the same thing. An example of the former:



It takes a little thought to convert this structure into the appropriate FOR-NEXT loops, although it's obvious after a little while that both the initialization value (e.g. K = 1) and the step (e.g. K = K + 1) are both present, as is the termination value (e.g. K > 25).

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Note that the Warnier diagram also uses the IF-THEN-ELSE which it implements like this:



If this check for those lousy zero quiz scores is added to the design and the code, then the Warnier will have to be modified so that the inside J-loop that was diagrammed in the previous section will serve as an explanation to the line that says PRINT ASTERISKS. In other words, the Warnier will grow to the right with this insertion.

Now that we've spent all this time fooling around with Warniers, which we've already admitted don't work all that well in practice, we come to the real crux of the matter. The only way to properly code a Warnier diagram is to make use of the SUBROUTINE! Each level to the right is actually a subroutine which is called from the point in the column that the brace points to. This means that the code for our original one column Warnier might look like this:

```
100 GOSUB 1000      ' get initial information
110 GOSUB 2000      ' open output file
120 GOSUB 3000      ' loop through quiz scores
130 GOSUB 4000      ' close file
140 END
```

Since subroutines 2000 and 4000 are going to be one line long, plus the identifying subroutine header (of course we use lots of remarks so we don't forget what all these subroutines do), plus the return statement, we might as well just put the information in lines 110 and 130. Those who are rabid about structured design, of course, would never think of such a thing. Keep in mind that in "real" programs, we frequently have a lot of work to do when opening and closing files, and a subroutine in these cases makes a lot of sense.

So why is all this use of subroutines such a boon for mankind? You can quickly glance at the driver portion of a program, with all the explanations for the subroutines,

Introducing the Creative Genius...

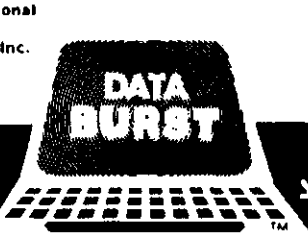
YOU. You have the system . . . and now you have the means. DataBurst™, a genuine full-screen processor with a BASIC program generator, brings mainframe technology to your IBM® -PC. The unique DataBurst™ screen editor allows you to design screen formats quickly and easily. Screen formats are program-independent, reducing both development and maintenance time.

During execution of your BASIC program, DataBurst™ will handle all of your screen and keyboard operations through one Assembly language module, which requires less than 12K of RAM. A true full-screen processor, DataBurst™ allows unlimited design complexity, turning every programmer into a 'creative genius'.

For your convenience, DataBurst™ is available through your local computer retailer or directly from Key Solutions, Inc. To order directly, please send check or money order for $225.00 to Key Solutions, Inc., P.O. Box 2297, Santa Clara, CA 95055 (California residents add applicable sales tax).

IBM® -PC is a registered trademark of International Business Machines Corporation.
DataBurst™ is a trademark of Key Solutions, Inc.

The Design Tool for the Creative Programmer



KEY SOLUTIONS

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

and virtually flowchart it in your head. The overall structure of the program becomes very obvious, almost trivial. As you begin to examine facets of the program, you know where to look for the details, and you keep descending through levels of nested subroutines until you get to the level of detail that you want. Many design bugs, for example, are easily caught because the LOGIC is not encumbered by the actual coding and overwhelming detail of variable names and punctuation.

Once the idea of using subroutines and Warniers is firmly implanted, the experienced programmer can do what every other lazy s.o.b. does, and skip the pencil and paper stage. The difference is that, with the subroutine method in mind, the programmer simply types out a program heading, then proceeds to type a whole series of GOSUBS with explanatory remarks. The result, in one pass, is a complete program outline. Then the programmer can go back and begin working on sections, constantly moving from the gross level to the detailed level.

Incidentally, my own rule for when to stop calling more and more subroutines and when to start writing lines of code, is that no subroutine shall be more than one page long (about 60 lines of printing, including line numbers that spill over onto more than one line), nor shall a subroutine be shorter than ten lines. This way the program listings can be broken up into separate pages that correspond to subroutines that are easily thumbed through. Of course, not liking zealotry, I do break my own rules. Virtually every time, I wind up with a subroutine that spills over to two or three pages. Subsequent analysis indicates that I've written another piece of trash.

So, while BASIC is often criticized for not having a subroutine statement that will accept parameters, like FORTRASH has, the subroutine is, in my opinion, the most useful concept to come out of either STRUCTURED PROGRAMMING or STRUCTURED DESIGN. Putting together structured programming and structured design means better programs for you and, if you tend to get paid by the job like I do, more money in your pocket. Amen. □

© 1983 Eugene Computer Designs

BE PROUD OF YOUR PROGRAMMING EXPERTISE!!

Share your special interest in computers at sporting events, parties, shopping, or wherever fun people meet. Wear the "OFFICIAL COMPUTER HACKER "T-Shirt. This quality shirt displays the 3 color "Hacker" logo for all to enjoy.

A perfect gift.

Send orders stating size (S, M, L, XL) to

Eugene Computer Designs
P.O. Box 48
Eugene, Oregon 97440

$9.95 each

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRINTER CONFIGURATION IN ASSEMBLY

by Rick Millhollin

Programs written in assembly language can be made quite readable if the author is careful about structure and includes appropriately named constants. Deciding to define a number of constants with variable names can make the difference between clear code and utter confusion. Use all of the tools a programming language offers, the ones to optimize the final execution speed as well as those designed to make your programs readable and maintainable.

The program is designed to set up the Okidata Microline 92 printer.

```
.xcref
.xall
.list
page    60,132
title   OKIDATA -- Set up Okidata Microline 92 Printer Options
comment \*************************************************************************

OKIDATA Version 1.00 -- Rick Millhollin -- 10/04/83

This program sets up the basic Okidata Microline 92 printer options:
        Character pitch,
        Line spacing,
        Printing mode and speed.
It must be used for this function rather than the DOS MODE command, due to the
incompatibility between control codes for the IBM (Epson) and Okidata printers

Notes:  (1) This program will work properly only on an 80-column monochrome
            display and with the printer installed as LPT1.
        (2) Standard Okidata Microline 92 control mode is assumed.  For a
            printer with the optional IBM Plug'n Play ROM, this program is not
            needed and the DOS MODE command can be used.

*************************************************************************\
;
; Macro and symbol definitions.
;
bioscall macro  type
        int     type            ; issue bios call int of appropriate type
        endm
;
doscall macro
        int     21h             ; issue dos call int
        endm
;
cr      equ     0dh             ; carriage return
lf      equ     0ah             ; line feed
null    equ     00h             ; null
space   equ     ' '             ; space
fullscr equ     80*25           ; num of chars on screen
display equ     10h             ; bios int type -- display
printer equ     17h             ;                  printer
positn  equ     02              ; bios function code -- position cursor
dispchr equ     10              ;                       display char
print   equ     00              ;                       print char
init    equ     01              ;                       init printer/get status
lpt     equ     00              ; lpt1 for bios calls
ready   equ     00h             ; printer status -- ready
dispstr equ     09              ; dos function code -- display string
read    equ     01              ;                      read/echo char
        page
;
; Stack segment.
;
STACK   SEGMENT para stack 'stack'
        db      256 dup(0)
stack   ends
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;
; Data segment.
;
DATA    SEGMENT para public 'data'
prnterr db      'Printer not ready',cr,lf,'$'
menu    db      'OKIDATA -- Version 1.00',cr,lf,lf
        db      'Okidata Microline 92 Printer Options:',cr,lf,lf
        db      '   (1)  160 Characters/Second Data Processing Quality',cr,lf
        db      '   (2)  40 Characters/Second Correspondence Quality',cr,lf
        db      '   (3)  10 Characters/Inch, 80 Columns',cr,lf
        db      '   (4)  12 Characters/Inch, 96 Columns',cr,lf
        db      '   (5)  17 Characters/Inch, 136 Columns',cr,lf
        db      '   (6)  6 Lines/Inch',cr,lf
        db      '   (7)  8 Lines/Inch',cr,lf
        db      '   (8)  Test Print',cr,lf
        db      '   (9)  Exit',cr,lf,lf
        db      'Enter desired option number: $'
optnpos db      29,14           ; option input cursor position -- col, row
optnmsg db      cr,lf,lf,'Option '
optnnum db      '# selected$'
optnerr db      cr,lf,lf,'Invalid option '
optnbad db      '# $'
ctlcode db      1bh,30h         ; esc-0 -- 160 cps data processing quality
        db      1bh,31h         ; esc-1 -- 40 cps correspondence quality
        db      1eh,null        ; rs -- 10 cpi
        db      1ch,null        ; fs -- 12 cpi
        db      1dh,null        ; gs -- 17 cpi
        db      1bh,36h         ; esc-6 -- 6 lpi
        db      1bh,38h         ; esc-8 -- 8 lpi
pattern db      cr,lf,'ABCDEFGHIJKLMNOPQRSTUVWXYZ',cr,lf
        db      'abcdefghijklmnopqrstuvwxyz',cr,lf
        db      '1234567890',cr,lf
        db      '!@#$%^&*()_-+={}[];:"',"'`~|<>?\/.,",cr,lf,null
data    ends
        page
;
; Code segment.
;
CODE    SEGMENT para public 'code'
OKIDATA PROC    far
;
; Standard DOS program prologue.
;
        assume  cs:code,ds:data,es:nothing,ss:stack     ; establish seg reg's
        push    ds              ; save psp seg addr
        xor     ax,ax           ; clear reg
        push    ax              ; save return offset in psp -- 0
        mov     ax,seg data     ; load data seg addr
        mov     ds,ax           ; mov to seg reg


;
; Initialize the printer and check status.
;
        mov     ah,init         ; load function code -- init/get status
        mov     dx,lpt          ; load printer location
        bioscall printer        ; call bios
        cmp     ah,ready        ; check if ready
        je      clear           ; yes -- go ahead
        mov     dx,offset prnterr       ; load string addr -- error msg
        mov     ah,dispstr      ; load function code -- display string
        doscall                 ; call dos
        ret                     ; return to dos
;
; Clear the screen and display the menu.
;
clear:  xor     dx,dx           ; clear reg -- position (0,0)
        mov     ah,positn       ; load function code -- position cursor
        bioscall display        ; call bios
        mov     al,space        ; load char to write -- space
        mov     cx,fullscr      ; load num of chars to write -- full screen
        mov     ah,dispchr      ; load function code -- display char
        bioscall display        ; call bios
        mov     dx,offset menu  ; load string addr -- option menu
        mov     ah,dispstr      ; load function code -- display string
        doscall                 ; call dos
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;
; Read and validate the option selection.
;
        jmp     short skip      ; skip cursor positioning first time
option: mov     dx,word ptr optnpos     ; load cursor position
        mov     ah,positn       ; load function code -- position cursor
        bioscall display        ; call bios
        mov     al,space        ; load char to write -- space
        mov     cx,1            ; load num of chars to write -- 1
        mov     ah,dispchr      ; load function code -- display char
        bioscall display        ; call bios
skip:   mov     ah,read         ; load function code -- read/echo
        doscall                 ; call dos
        cmp     al,'1'          ; check if option < 1
        jb      badoptn         ; yes -- go report error
        cmp     al,'8'          ; check if option <= 8
        jbe     ok              ; yes -- go ahead
        cmp     al,'9'          ; check if option = 9
        je      exit            ; yes -- go exit
badoptn: mov    optnbad,al      ; move option to msg
        mov     dx,offset optnerr       ; load string addr -- error msg
        mov     ah,dispstr      ; load function code -- display string
        doscall                 ; call dos
        jmp     option          ; go try again
;
; Write the ok message and check for test print.
;
ok:     mov     optnnum,al      ; move option to msg
        mov     bl,al           ; save option
        mov     dx,offset optnmsg       ; load string addr -- ok message
        mov     ah,dispstr      ; load function code -- display string
        doscall                 ; call dos
        cmp     bl,'8'          ; check if digit = 8
        je      test            ; yes -- go test print
;
; Write the control code(s) to the printer.
;
        xor     bh,bh           ; clear hi reg
        sub     bx,'1'          ; compute option offset from 1
        shl     bx,1            ; compute control code offset in table
        mov     al,[ctlcode+bx] ; load control code
        mov     ah,print        ; load function code -- print char
        mov     dx,lpt          ; load printer location
        bioscall printer        ; call bios
        inc     bx              ; inc char addr
        mov     al,[ctlcode+bx] ; load control code
        cmp     al,null         ; check for null -- no second char
        je      option          ; yes -- go for another option
        mov     ah,print        ; load function code -- print char
        bioscall printer        ; call bios
        jmp     option          ; go for another option
; Print a small test pattern.
;
test:   cld                     ; set forward direction
        mov     si,offset pattern       ; load pattern addr
loop:   lodsb                   ; load char
        cmp     al,null         ; check for end
        je      option          ; yes -- exit loop and go for another option
        mov     ah,print        ; load function code -- print char
        mov     dx,lpt          ; load printer location
        bioscall printer        ; call bios
        jmp     loop            ; go do it again
;
; Clear screen and exit program.
;
exit:   xor     dx,dx           ; clear reg -- position (0,0)
        mov     ah,positn       ; load function code -- position cursor
        bioscall display        ; call bios
        mov     al,space        ; load char to write -- space
        mov     cx,fullscr      ; load num of chars to write -- full screen
        mov     ah,dispchr      ; load function code -- display char
        bioscall display        ; call bios
        ret                     ; return to dos
okidata endp
code    ends
        end     okidata     □
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PATCH

THE DOS PATCH UTILITY

by Brett Salter, author of PEEKS n' POKES and the Inside Track

When IBM announces a new version of DOS, are you faced with the problem of converting many diskettes to the new release? Do you lose room on your diskettes to the three IBM system files (IBMBIO.COM, IBMDOS.COM, and COMMAND.COM)? If the answer to either of the above questions is YES, I have something that can help you, providing you have a RAM disk or a fixed disk.

The solution is to make up one DOS disk that contains the three files mentioned above. Always use this disk and only this disk to boot the system. Remove the three system files from your other disks. After boothing, the files IBMBIO.COM and IBMDOS.COM are not used again. COMMAND.COM however, is different. Part of COMMAND.COM is loaded into high membory and can be overwritten by your programs; it then becomes unavailable to the system. DOS always wants to reload COMMAND.COM from drive A, forcing you to insert a DOS diskette. The programs decribed below will allow you to patch COMMAND.COM so that it will try to reload itself from another drive (specifically drive C), such as your RAM or fixed disk. If COMMAND.COM cannot be found on drive A, you'll be prompted to insert a disk containing COMMAND.COM in drive A.

To apply the patches to COMMAND.COM, I've written a general purpose patching program. This program eliminates the need to use DEBUG to modify a program. The patching program, PATCHER.BAS, reads a data file containing the name of the file to be patched, a checksum, comments, and the actual patches. Each patch "record" consists of the offset into the file, the old value for the byte, and the new value for the byte. Since the program verifies the old value and the checksum, you can be sure that the patches you desire are correctly applied to the program.

There are two data files — COMCOM11.PAT is used to apply the patches for DOS 1.10 and COMCOM20.PAT is used to apply the patches for DOS 2.00.

The procedure for applying the patches is as follows:

1) Boot the system.
2) Backup COMMAND.COM.
3) Key in PATCHER.BAS and the appropriate version of COMCOMxx.PAT on drive A (the default drive).
4) Enter "BASIC PATCHER" and press return to start the patch program.
5) Respond with either "COMCOM11.PAT" or "COMCOM20.PAT" when prompted for the patch file.
6) If errors occur, check PATCHER.BAS, COMCOMxx.PAT, and the patches for version 1.10 or version 2.00.

After correcting any errors, restore COMMAND.COM from the backup copy, and go back to step 4.

7) If no errors occur, change your AUTOEXEC.BAS to include a "COPY A:COMMAND.COM = C:COMMAND.COM" and reboot the system to load the patched version of COMMAND.COM.
8) Test the patch by running a program that will overwrite the transient portion of COMMAND.COM (a compiled BASIC program will do this). When you return to DOS, note that drive A stays still! Experiment by deleting COMMAND.COM from drive C and running the compiled BASIC program again. This time drive A will be used.

This patch does have an Achilles heel— if the version of COMMAND.COM found on drive C is not what DOS wants, (e.g. the version of COMMAND.COM on drive C is not compatible with DOS) the system will display the "invalid COMMAND.COM" message and prompt you to insert the correct version. Since you can't change the disk on drive C when it's a RAM disk or a fixed drive, you will have to reboot the system. The way to avoid this problem is to be sure to copy COMMAND.COM to drive C as noted in step 7, above.

If you want to use a drive other than C, change the patch files as follows:

For COMCOM11.PAT — change the tenth line in COMCOM11.PAT from "660, 100, 3" to:
"660, 100, 2" and decrement the checksum by 1 for drive B
"660, 100, 4" and increment the checksum by 1 for drive D

For COMCOM20.PAT — change the fifth line in COMCOM20.PAT from "100, 0, 67" to:
"100, 0, 66" and decrement the checksum by 1 for drive B
"100, 0, 68" and increment the checksum by 1 for drive D

If you've modified a program and want to pass the patches along, you can use the program GENPATCH. BAS to generate a patch file for use by PATCHER.BAS. GENPATCH compares the original version of the program with your patched version and generates the patch file. If possible, run GENPATCH using BASIC 2.00 —the lack of file buffering prior to DOS 2.00 can make this program incredibly slow. If DOS 2.00 is not available, use a RAM drive.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

For BASIC 1.10 and prior version, both PATCHER.BAS and GENPATCH.BAS are limited to patching the first 32,767 bytes in the file. With BASIC 2.00 the limit is so large that it is not a potential problem.

When DOS 2.10 and/or 3.00 become available, I'll work up the patches to make COMMAND.COM behave like it should. That is, of course, if Microsoft or IBM don't fix the problem by making COMSPEC work!

```
a:command.com, 28735 ,"Reload COMMAND.COM from drive C - DOS 1.10 version"
 323 , 205 , 233
 324 , 33 , 77
 325 , 10 , 1
 326 , 192 , 144
 327 , 116 , 144
 328 , 14 , 144
 658 , 97 , 36
 659 , 110 , 176
 660 , 100 , 3
 661 , 32 , 162
 662 , 115 , 40
 663 , 116 , 4
 664 , 114 , 205
 665 , 105 , 33
 666 , 107 , 10
 667 , 101 , 192
 668 , 32 , 116
 669 , 97 , 14
 670 , 110 , 176
 671 , 121 , 1
 672 , 32 , 162
 673 , 107 , 40
 674 , 101 , 4
 675 , 121 , 205
 676 , 32 , 33
 677 , 119 , 10
 678 , 104 , 192
 679 , 101 , 116
 680 , 110 , 3
 681 , 32 , 233
 682 , 114 , 157
 683 , 101 , 254
 684 , 97 , 233
 685 , 100 , 168
 686 , 121 , 254
```

```
a:command.com, 18759 ,"Reload COMMAND.COM from drive C -- DOS 2.00 version"
 97 , 0 , 139
 98 , 0 , 218
 99 , 0 , 176
 100 , 0 , 67
 101 , 0 , 136
 102 , 0 , 7
 103 , 0 , 176
 105 , 0 , 205
 106 , 0 , 33
 107 , 0 , 115
 108 , 0 , 14
 109 , 0 , 176
 110 , 0 , 65
 111 , 0 , 136
 112 , 0 , 7
 113 , 0 , 184
 115 , 0 , 61
 116 , 0 , 205
 117 , 0 , 33
 118 , 0 , 115
 119 , 0 , 3
 120 , 0 , 233
 121 , 0 , 249
 122 , 0 , 4
 123 , 0 , 233
 124 , 0 , 6
 125 , 0 , 5
 126 , 0 , 144
 127 , 0 , 144
 128 , 0 , 144
 169 , 128 , 96
 393 , 128 , 96
 732 , 128 , 96
1392 , 205 , 233
1393 , 33 , 238
1394 , 115 , 250
1395 , 16 , 144
2946 , 128 , 96
```

```
10 ' PATCHER - file patching program
20 ' Copyright 1983 - Data Base Decisions, Atlanta, GA
30 ' This program is used to patch other programs or files. It requires
40 ' a data file containing the patches. The first three items in the
50 ' patch file are the name of the file to be patched, a checksum, and
60 ' comments. For each byte to be patched, there is one record containing
70 ' the offset of the byte to be patched, the old value of the byte,
80 ' and the new (patch) value.
90 ' Note: If the offset is greater than 32,767, BASIC 2.00 must be used.
100 DEFINT A-Z
110 CLEAR
120 ON ERROR GOTO 460
130 CLOSE
140 PRINT:INPUT "Name of file containing patches";PAT$
150 IF PAT$="" THEN END
160 OPEN "i",#1,PAT$
170 INPUT #1,FIL$,CKSUM!,COMMENT$
180 OPEN "i",#2,FIL$ ' make sure it is there
190 PRINT "Patching: " FIL$
200 PRINT "Comments: " COMMENT$
210 CLOSE 2
220 OPEN "r",#2,FIL$,1 ' reopen as random file
230 FIELD 2,1 AS R$
240 IF EOF(1) THEN 410
250 INPUT# 1,BYTE!,OLDVAL,NEWVAL ' get patch
260 NEWSUM!=(NEWSUM!+BYTE!+OLDVAL+NEWVAL)
270 IF NEWSUM!)32767 THEN NEWSUM!=NEWSUM!-32767:GOTO 270
280 IF BYTE!)LOF(2) THEN PRINT "Byte" BYTE! "is beyond end of file":GOTO 360
290 GET 2,BYTE!
300 R=ASC(R$)
310 IF R()OLDVAL THEN PRINT "Old value for byte" BYTE! "is" R "not" OLDVAL
                                                              :GOTO 360
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
320 LSET R$=CHR$(NEWVAL)
330 PUT 2,BYTE!
340 APPLIED=APPLIED+1
350 GOTO 240
360 REM *** invalid condition ***
370 BEEP:INPUT "Continue (y/n)";ANS$
380 IF ANS$="Y" OR ANS$="y" THEN 350
390 IF ANS$="N" OR ANS$="n" THEN 410
400 GOTO 360
410 REM *** wrap it up ***
420 IF CKSUM!=NEWSUM! THEN PRINT "Checksums match" ELSE PRINT
                        "Checksums do not match -- input value is" CKSUM! "and
                         calculated value is" NEWSUM!:BEEP
430 PRINT "Patches applied: " APPLIED
440 CLOSE
450 END
460 REM *** error handler ***
470 UNABLE$="Unable to "
480 IF ERL=160 OR ERL=250 THEN PRINT UNABLE$ "read " PAT$
490 IF ERL=180 OR ERL=220 OR ERL=290 THEN PRINT UNABLE$ "read " FIL$
500 IF ERL=330 THEN PRINT UNABLE$ "write " FIL$
510 RESUME 10
```

```
10 ' GENPATCH - generate patches for PATCHER.BAS
20 ' Copyright 1983 - Data Base Decisions, Atlanta, GA
30 ' This program is used to generate a patch file to be used by PATCHER.BAS.
40 ' It reads two versions of a program and generates the differences to
50 ' a temporary patch file named GENPATCH.TMP. After the two versions of
60 ' the program have been read, GENPATCH.TMP is copied to the patch file
70 ' you specify, adding the file name, checksum, and comments as the first
80 ' record.
90 ' Note: If the offset is greater than 32,767, BASIC 2.00 must be used.
100 ' Due to the lack of disk buffering in BASIC 1.10 and prior versions,
110 ' DOS 2.00 should be used. If this is not possible, use a RAM disk
120 ' for the best performance.
130 DEFINT A-Z
140 CLEAR
150 ON ERROR GOTO 510
160 INPUT "Old version of file";OFIL$
170 OPEN "i",#1,OFIL$ ' check to be sure it is there
180 CLOSE 1
190 OPEN "r",#1,OFIL$,1 ' open as random
200 FIELD 1,1 AS O$
210 INPUT "New version of file";NFIL$
220 OPEN "i",#2,NFIL$ ' check to be sure it is there
230 CLOSE 2
240 OPEN "r",#2,NFIL$,1 ' open as random
250 FIELD 2,1 AS N$
260 INPUT "File in which to save patches";DIF$
270 TMP$="GENPATCH.TMP"
280 OPEN "o",#3,TMP$ ' write to a temp file
290 FOR BYTE!=1 TO LOF(1) ' compare the bytes
300 GET 1,BYTE!
310 GET 2,BYTE!
320 IF O$=N$ THEN 380 ' they are equal
330 OLDVAL=ASC(O$) ' convert to ascii
340 NEWVAL=ASC(N$)
350 PRINT# 3, BYTE! "," OLDVAL "," NEWVAL
360 NEWSUM!=NEWSUM!+BYTE!+OLDVAL+NEWVAL ' keep checksum
370 IF NEWSUM!)32767 THEN NEWSUM!=NEWSUM!-32767:GOTO 370
380 NEXT BYTE!
390 CLOSE
400 OPEN "i",#1,TMP$ ' copy temp file to patch file
410 OPEN "o",#2,DIF$
420 PRINT "Comments to be put in " DIF$;:INPUT "";COMMENT$
430 PRINT #2,NFIL$ "," NEWSUM! "," CHR$(34) COMMENT$ CHR$(34) ' add record
440 IF EOF(1) THEN 480
450 LINE INPUT# 1, X$
460 PRINT# 2, X$
470 GOTO 440 ' copy another record across
480 CLOSE
490 KILL TMP$ ' delete temporary file
500 END
510 REM *** error handler ***
520 UNABLE$="Unable to "
530 IF ERL=170 OR ERL=190 OR ERL=300 THEN PRINT UNABLE$ "read " OFIL$
540 IF ERL=220 OR ERL=240 OR ERL=310 THEN PRINT UNABLE$ "read " NFIL$
550 IF ERL=280 OR ERL=350 THEN PRINT UNABLE$ "write " TMP$
560 IF ERL=400 OR ERL=450 THEN PRINT UNABLE$ "read " TMP$
570 IF ERL=410 OR ERL=460 THEN PRINT UNABLE$ "write " DIF$
580 IF ERL=490 THEN PRINT UNABLE$ "delete " TMP$
590 RESUME 10
```

□

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SWIG©

SOFTWARE WRITERS INTERNATIONAL GUILD

SCHEDULED SWIG ACTIVITIES & MEMBERSHIP BENEFITS

(1) $10,000 PROGRAMMING CONTEST (Members only)

(2) NATIONAL COMPUTER WEEK

(3) ANNUAL CONFERENCE AND SOFTWARE AWARDS CEREMONY (During National Computer Week)

(4) CONSULTANT REGISTRY (With computer store referral system for customized software)

(5) JOB PLACEMENT SERVICE (Free to individual members, fixed maximum fee to companies)

(6) FREE SEMINARS & MEETINGS LOCALLY

(7) SOFTWARE LIBRARY LENDING & EXCHANGE SERVICE (Professional quality assemblers, utilities, games, etc.)

(8) SOFTWARE LOCATION SERVICE (For companies & individuals-if it exists, **SWIG** will find it. If not, see #9)

(9) SOFTWARE DEVELOPMENT SERVICE (From novice to scientist, **SWIG** members can work on any project-from applications to games to R&D)

(10) LEGAL SERVICE

(11) AGENT (**SWIG** can represent you in sales to software publishers)

(12) 24 HOUR - 7 DAY BULLETIN BOARD SYSTEM (BBS) ACCESSIBLE BY COMPUTER FREE

(13) AND MORE!!!!

THE LARGEST PAID MEMBERSHIP PROGRAMMERS GUILD - OVER 5,000 MEMBERS WORLDWIDE!!

MEMBERSHIP APPLICATION FOR SOFTWARE WRITERS INTERNATIONAL GUILD

NAME _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

PHONE # ( ) _____

- CLASSIFICATION:

  ☐ NOVICE ☐ BEGINNER TO ADVANCED

  ☐ ADVANCED WITH ON THE JOB EXPERIENCE ☐ RESEARCH/SCIENTIST

- WHAT EQUIPMENT DO YOU HAVE EXPERIENCE WITH &/OR ACCESS TO &/OR PLAN TO BUY?

  ☐ MAINFRAME ☐ MINI ☐ MICRO ☐ DESIGN/R&D

  BRAND NAME(S): ☐ IBM ☐ XEROX ☐ APPLE ☐ TI

  ☐ COMMODORE ☐ RADIO SHACK ☐ ATARI ☐ OSBORNE

  ☐ TIMEX/SINCLAIR ☐ NORTH STAR ☐ HEWLETT PACKARD

  ☐ OTHER _____

- AREAS OF INTEREST:

  ☐ DATA PROCESSING ☐ BUSINESS APPLICATIONS ☐ GRAPHICS

  ☐ LEGAL ☐ VOICE ☐ MEDICAL ☐ APPLIANCE (HOME) CONTROL

  ☐ ROBOTICS ☐ GAMES ☐ MUSIC ☐ R&D ☐ OTHER _____

- MEMBERSHIP ACTIVITIES AND SERVICES OF INTEREST:

  READ THE LIST ON THE LEFT AND CIRCLE THE NUMBERS BELOW THAT APPLY.

  1 2 3 4 5 6 7 8 9 10 11 12

☐ I HAVE ENCLOSED $20 ANNUAL MEMBERSHIP FEE ☐ CK ☐ MO
(MAKE CHECK PAYABLE TO: **SWIG**)

RETURN TO: **SWIG**
P.O. BOX 87
STONY POINT, NEW YORK 10980
(914) 354-5585

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C THE DIFFERENCE

by Ed Keating

If you keep up with programming languages and have read the September issue of *PC magazine* (pp. 128-142), you may have seen the benchmark programs comparing several languages.

Most of my favorite, C, was omitted yet the time results of it's run were reported, albeit erroneously. Here we present the missing benchmarks in C, with its counterpart in BASIC and the results of both as run.

Don't let C's newness (in popularity) put you off. It is much more common-sensical than FORTRAN or RATFOR (RATional FORtran) and less strict than PASCAL. Basic programmers will take a while to adjust to the lack of line numbers — but once they're free of that dependence, they should find C very similar.

The two programs reprinted here do the same thing.

One is a BASIC program consisting of a main section (lines 100-140), six subroutines and a data section. The main C program, similarly structured, is followed by a collection of modules that perform various duties. The main program in C has the identical function as the BASIC lines 100-140, the statements following the opening brace { are calls (gosubs) to the individual modules. Modules are invoked sequentially until the closing brace } is reached,

```
10 DEFINT X,L : DEFDBL A,B,C
20 DIM Z(25)
100 GOSUB 1000
110 GOSUB 2000
120 GOSUB 3000
130 GOSUB 4000
140 GOSUB 5000
599 STOP
600 DEF SEG=&H40 :' Time difference good for up to 1 hour.
610 T = PEEK(&H6C) + PEEK(&H6D)*256
620 T = T * 55
630 PRINT "Time difference was ";T;" Milliseconds"
640 DEF SEG
650 RETURN
1000 PRINT "Empty loop in basic"
1001 TIME$ = "0"
1010 FOR X = 1 TO 10000
1020 NEXT
1030 GOSUB 600
1040 RETURN
1050 REM
2000 PRINT "Integer count in Basic"
2001 TIME$ = "0"
2005 X = 0
2010 WHILE X < 32767
2020 X = X + 1
2025 WEND
2030 GOSUB 600
2040 RETURN
3000 PRINT "Floating point benchmark"
3001 TIME$="0"
3005 A = 0: B = 1234.56 : C = 78.9
3010 FOR X = 1 TO 10000
3020 A = B*C
3030 A = B/C
3040 NEXT
3050 GOSUB 600
3060 RETURN
4000 PRINT "String concatenation test "
4001 TIME$="0"
4010 FOR X = 1 TO 10000
4020 A$="This is a string"
4030 B$="This is a longer string with lots of word in it"
4040 C$ = A$ + B$
4050 NEXT
4060 GOSUB 600
4070 RETURN
5000 PRINT"Table lookup test"
5010 TIME$="0"
5020 FOR X = 1 TO 1000
5030  RESTORE 6000
5040  FOR L = 1 TO 25
5050   READ Z(L)
5060  NEXT L
5070 NEXT
5080 GOSUB 600
5090 RETURN
6000 DATA 1,2,3,4,5
6010 DATA 6,7,8,9,10
6020 DATA 11,12,13,14,15
6030 DATA 16,17,18,19,20
6040 DATA 21,22,23,24,25
9999 END
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
#include <stdio.h>
main(argc,argv)         /* timestamp program using Lattice C */
int argc;
char *argv[];
{
        looptest(); /* perform simple loop test */
        addints();  /* add integers test */
        fptest();   /* floating point test */
        strings();  /* string concatenation */
        tables();   /* table lookup test */
        printf("End benchmarks\n");
}
printimes(t1,t2)        /*print timestamps on standard output */
int t1[8],t2[8];        /* timestamp buffers */
{
        long d,tdiff(); /* for time difference calculations */
        char tsbuf[20]; /* used to format time info */
        char *ftime();  /* formats a timestamp into a buffer */

        printf("Time started was %s\n",ftime(t1,tsbuf));
        printf("Time ended   was %s\n",ftime(t2,tsbuf));
        d = tdiff(t1,t2);
        printf("The difference in times in Milliseconds is %ld\n",d);
} /* end of printimes*/
looptest()      /* basic loop test */
{
        int i,t1[8],t2[8];      /* timestamp buffers */

        gtime(t1);      /* get time one. */
        for(i =0; i < 10000; i++);
        gtime(t2);
        printf("\nSimple loop test\n");
        printimes(t1,t2);
}
addints()       /* add integers test */
{
        int i,t1[8],t2[8];      /* timestamp buffers */
        printf("\nAdding integers loop to 32767\n");
        gtime(t1);
        i = 0;
        while(i < 32767) i++;
        gtime(t2);
        printimes(t1,t2);
}
fptest()        /* floating point test */
{
        int i,t1[8],t2[8];      /* timestamp buffers */
        double  a,b,c;

        printf("\nFloating point benchmark\n");
        gtime(t1);
        a = 0; b = 1234.56; c = 78.9;
        for(i = 0; i < 10000; i++) {
         a=b*c;
         a=b/c;
        }
        gtime(t2);
        printimes(t1,t2);
}
strings()
{
        int i,t1[8],t2[8];      /* timestamp buffers */
        char *pa,*pb,*pc;
        static char sa[] = "This is a string";
        static char sb[] = "This is a longer string with lots of words in it";
        char sc[100];

        printf("\nString concatenation benchmark\n");
        gtime(t1);
        for(i = 0; i < 10000; i++) {
         pc = &sc[0];
         pa = &sa[0];
         pb = &sb[0];
         while(*pc++ = *pa++); /* copy up to first null */
         pc--;   /* back up over trailing null */
         while(*pc++ = *pb++);   /* add in the second string */
        }
        gtime(t2);
        printimes(t1,t2);
}
```

Continued on page 8

at which point there is a RETURN.

There are other similarities between the two programs. The C semicolon ; can be said to be equivalent to the BASIC colon :, both being used to separate statements. (In C, however, the semicolon is essential for ending a statement, whereas in BASIC line ends have that effect by default.) As in BASIC CALL routines, the comma , is used to separate expressions passed to functions. Unlike basic the equal sign = has only one meaning in C, namely an assignment operator. To perform a conditional test IF X = Y a double equal sign = = is used IF XX = = YY.

In C the open and close braces { , } define a complex statement. They may be used to control program flow in much the same way parentheses are used to control arithmetic execution. In C a complex statement can be written, anywhere a single statement can. Data types available in C program are the integer, long integer (32 bits), character, float, double, and Boolean. Long integer calculations can be made to run much faster than the equivalent floating point expressions used by BASIC and still retain the necessary precision. In C, a string constant is distinguished from a single character constant by using double quotes " for the former and single quotes ' for the latter.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
tables()        /* table lookup tests */
{
        int i,j,t1[8],t2[8];        /* timestamp buffers */
        static int data[25] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
                21, 22, 23, 24, 25 };
        int array[25];

        printf("\nTable lookup\n");
        gtime(t1);
        for(i = 0; i < 1000; i++)
         for(j = 0; j < 25; j++) array[j] = data[j];
        gtime(t2);
        printimes(t1,t2);
}
```

```
Empty loop in basic
Time difference was  220  Milliseconds
Integer count in Basic
Time difference was  880  Milliseconds
Floating point benchmark
Time difference was  15895  Milliseconds
String concatenation test
Time difference was  19635  Milliseconds
Table lookup test
Time difference was  20515  Milliseconds
```

```
A>gtest

Simple loop test
Time started was  0: 1:46.390
Time ended   was  0: 1:46.610
The difference in times is 220 Milliseconds

Adding integers loop to 32767
Time started was  0: 1:47.320
Time ended   was  0: 1:48. 30
The difference in times is 710 Milliseconds

Floating point benchmark
Time started was  0: 1:48.690
Time ended   was  0: 2:55.480
The difference in times is 66790 Milliseconds

String concatenation benchmark
Time started was  0: 2:56.200
Time ended   was  0: 3:20.360
The difference in times is 24160 Milliseconds

Table lookup
Time started was  0: 3:21. 20
Time ended   was  0: 3:21.900
The difference in times is 880 Milliseconds
End benchmarks
```

The standard module for output, PRINTF, is somewhat like a super BASIC *PRINT USING*. The first parameter is a string constant which defines the template in which the percent **%** character is used as Escape, to describe what kind of formatting is available. The C programmer can choose from octal, decimal, hexadecimal with or without long integers, strings and special characters which are introduced with the backslash **/**. The character constant of **/n** is known to all C programmers as the newline character, the same result generated by the BASIC print when you DON'T end the line with a semicolon.

Basic programmers will take a while to adjust to the lack of line numbers

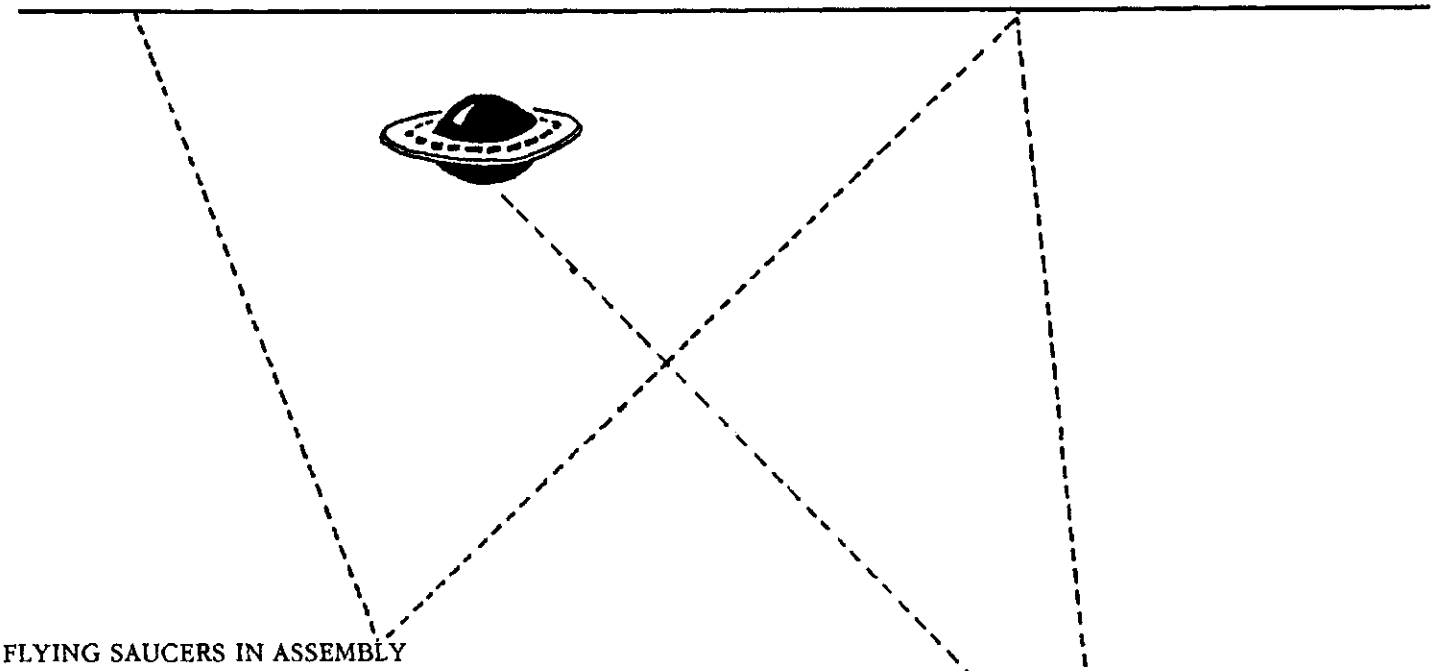
There are several good texts on C available, The one that defines the language is *The C Programming Language* by Brian Kernighan and Dennis Ritchie. Another good book for novices is Thomas Plum's *Learning to Program in C*. But experience is the best teacher. Try using the examples offered here and see how much you can learn from your own comparison.

Edward Keating is a staff software engineer for Motorola, Inc. who is currently working on a portable terminal network for IBM. He holds a Bachelors in Math from Benedictine. He is a member of the NIIBMPCUG.

excerpt from Port Fol I/O, the newsletter of the Northern Illinois IBM PC Users group, 111 East Chestnut St. Chicago, IL 60611 □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SAUCERS



FLYING SAUCERS IN ASSEMBLY

by Dan Rollins

Here are a pair of assembler listings that work together to move a small shape around the screen. The SHAPE.ASM program is a shape-drawing tool that can be the kernel for many different arcade-style games. Just pass it an X,Y coordinate pair, a pointer to some shape data, and a function code and off you go. This routine will either draw a shape, or XOR the shape data into screen memory. The latter option may be the most useful because

1) it returns a flag indicating if the shape collided with anything and
2) it can be used to erase a shape as well as draw it.

If the shape already exists at the current location, then invoking the XOR function will erase it from the screen. The routine can only draw the shape on a 4-dot horizontal boundary because it works with bytes rather than individual dots. This limitation is a minor one.

Notice that when the module converts the X,Y coordinate to a screen address, it uses a sequence of shifts and adds to perform a pseudo-multiplication by 80. This is considerably faster than using the MUL or IMUL opcode.

The second listing, SAUCER.ASM is an example driver that demonstrates how to use the SHAPE module. It also shows how to set up a shape-table that defines several "states" for the shape. In this example, the outline of the saucer stays the same, but a red flash "travels" across the center of the saucer as it moves around the screen. This concept is the basis of all animation—a walking man can be represented by as few as 4 shapes, each with the legs and arms in a slightly different position.

The flow of logic is straightforward: the shape is erased, moved, and redrawn. Then a delay routine is called to keep the saucer from shooting across the screen as a blur. Change the value in the DELAY_COUNT variable to speed it up or slow it down. The program also checks the keyboard to see if you want to exit. As an exercise of your assembly language skill, try adding a routine that increases or decreases the speed of the flight according to which key is pressed.

Pay attention to the GROUP, SEGMENT, PUBLIC, and EXTRN declarations in the listings. This format allows any number of object modules to be pre-assembled and linked to a main program. The external modules can have their own segments, but the code must reside in a segment that is included in CODE_GRP. This make the library procedures accessible with a near call.

To run the SAUCER program, you must assemble both modules and then link them with:

A >LINK saucer + shape

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;-------------------------------------------------------------
; SHAPE  near library procedure
; Mid-res shape-drawing routine.  XORs or draws the shape.
; On entry: AH    = function code.  0 = XOR, 1 = draw (fastest)
;           DH    = line to draw shape (0-199)
;           DL    = column (0-79)
;           DS:SI = address of a shape-table entry in this format:
;                   si+0 = count of horizontal bytes (4 dots per byte)
;                   si+1 = count of vertical lines
;      si+2 through si+n = data to be transferred to video memory
;
;On exit: the shape is displayed.
;         if function is XOR (AH = 0) then
;           AH returns collision flag: 0 = no collision
;         AX destroyed.  All other registers preserved.

          public  shape                   ;standard
code_grp  group   lib_seg                 ; opening for
lib_seg   segment byte public 'code'      ;  library
          assume  cs:code_grp             ;   modules

shape     proc    near
          push    bx                ;all registers are used,
          push    cx                ; so they must be saved, except
          push    dx                ;  AX which returns a value
          push    bp
          push    di
          push    si
          push    es

;------- calculate address in video memory from DH,DL (line,column reference)
          push    ax                ;save function code

          mov     al,dh
          sub     ah,ah             ;AX = requested line number
          and     ax,0FFFEH         ;mask off even/odd bit (mult * 2)
          shl     ax,1              ; * 4
          shl     ax,1              ; * 8
          shl     ax,1              ; * 16
          mov     di,ax             ; save partial result
          shl     ax,1              ; * 32
          shl     ax,1              ; * 64
          add     di,ax             ; DI = line number * 80
          test    dl,1              ;line in odd bank?
          jz      line_ok           ; no, skip
          add     di,2000H          ; yes, add odd-bank offset
line_ok:
          sub     ah,ah
          mov     al,dl             ;add in the horizontal offset
          add     di,ax             ;DI = screen address to alter
          mov     ax,0B800H
          mov     es,ax             ;ES:[DI] is destination


;------- set up horizontal and vertical loop counters
          lodsb                     ;fetch the count of horizontal bytes
          sub     ah,ah
          mov     cx,ax             ; and place it in CX

          lodsb                     ;fetch the count of vertical lines
          mov     dx,ax             ; and place it in DX
          inc     dx                ;prime for first pass

          pop     ax                ;fetch function code
          cmp     ah,0              ;is XOR function requested?
          je      xor_1             ;  yes, select the XOR loop
                                    ;  else, just move the bytes directly
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;-------- transfer the bytes of the shape directly to the screen (fastest)
draw_1:
          dec     dx                ;done?
          jz      shape_exit        ; yes, exit to caller
          mov     bx,cx             ; no, continue.  Save counter
          mov     bp,di             ;  save destination
         rep movsb                  ;transfer the data directly to screen
          mov     di,bp
          mov     cx,bx             ;restore count, destination
          cmp     di,2000H          ;was this an odd line?
          jae     draw_2            ; yes, make odd adjustment
          add     di,2000H          ; no, point to odd line, below current
          jmp     draw_1            ;     and loop back for next line
draw_2:
          sub     di,2000H-80       ;point to even line, below current
          jmp     draw_1            ; and loop back for next

;-------- XOR the bytes and test for collision (slower than direct draw)
xor_1:
          dec     dx                ;done?
          jz      shape_exit        ; yes, exit to caller
          mov     bx,cx
          mov     bp,di             ;save count, destination
xor_2:
          lodsb
          test    es:[di],al        ;did it hit anything?
          jz      xor_3             ; no, skip
          mov     ah,-1             ; yes, set collision flag to true
xor_3:
          xor     es:[di],al        ;XOR (display or erase the dots)
          inc     di                ;point to next screen location
          loop    xor_2             ;and repeat for each horizontal byte

          mov     di,bp
          mov     cx,bx             ;restore count, destination
          cmp     di,2000H          ;was this an odd line?
          jae     xor_4             ; yes, make odd adjustment
          add     di,2000H          ; no, point to odd line below current
          jmp     xor_1             ;     and loop back for next line
xor_4:
          sub     di,2000H-80       ;point to even line below current
          jmp     xor_1             ; and loop back for next

shape_exit:
          pop     es                ;restore regs for exit
          pop     si
          pop     di
          pop     bp
          pop     dx
          pop     cx
          pop     bx
          ret                       ;<---- SHAPE procedure exit --------
shape     endp

lib_seg   ends
          end
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;----------------------------------------
; SAUCER  program
; Illustrates how to use the SHAPE library module.
; Animates a flying saucer, bouncing it off of the
;  edges of the screen.  Exit to DOS by pressing any key.
; External calls: SHAPE

stack_seg segment para stack
          db      256 dup(?)
stack_seg ends

data_seg segment para public 'data'
delay_count dw   20000          ;variable delay between frames
shape_data label byte
saucer_0 db  3                  ;data for when cycle = 0
         db  7                  ;
         db  000H, 03CH, 000H   ;             W W
         db  003H, 0FFH, 0C0H   ;          W W W W W W
         db  03FH, 00AH, 0FCH   ;       W W W     r r W W W
         db  0A0H, 0A0H, 0A0H   ;    r r     r r     r r
         db  03FH, 00AH, 0FCH   ;       W W W     r r W W W
         db  003H, 0FFH, 0C0H   ;          W W W W W W
         db  000H, 03CH, 000H   ;             W W

saucer_1 db  3                  ;data for when cycle = 1
         db  7
         db  000H, 03CH, 000H   ;             W W
         db  003H, 0FFH, 0C0H   ;          W W W W W W
         db  03FH, 082H, 0FCH   ;       W W W r     r W W W
         db  028H, 028H, 028H   ;       r r     rr      r r
         db  03FH, 082H, 0FCH   ;       W W W r     r W W W
         db  003H, 0FFH, 0C0H   ;          W W W W W W
         db  000H, 03CH, 000H   ;             W W

saucer_2 db  3                  ;data for when cycle = 2
         db  7                  ;
         db  000H, 03CH, 000H   ;             W W
         db  003H, 0FFH, 0C0H   ;          W W W W W W
         db  03FH, 0A0H, 0FCH   ;       W W W r r     W W W
         db  00AH, 00AH, 00AH   ;         r r     r r     r r
         db  03FH, 0A0H, 0FCH   ;       W W W r r     W W W
         db  003H, 0FFH, 0C0H   ;          W W W W W W
         db  000H, 03CH, 000H   ;             W W

saucer_3 db  3                  ;data for when cycle = 3
         db  7                  ;
         db  000H, 03CH, 000H   ;             W W
         db  003H, 0FFH, 0C0H   ;          W W W W W W
         db  03FH, 028H, 0FCH   ;       W W W   r r   W W W
         db  082H, 082H, 082H   ;    r      r r     r r     r
         db  03FH, 028H, 0FCH   ;       W W W   r r   W W W
         db  003H, 0FFH, 0C0H   ;          W W W W W W
         db  000H, 03CH, 000H   ;             W W
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;------- variables control position and motion of saucer
cur_xy    label   word          ;get a handle on the following two bytes
cur_x     db      39            ;start in center of screen
cur_y     db      99            ;
cycle     db      0             ;start with shape 0
x_vect    db      -1            ;start by moving left by 1 byte (4-dots)
y_vect    db      4             ;          and down by 4-lines

s_ptrs    dw      saucer_0      ;mini-table for quick access to
          dw      saucer_1      ; the four different saucer shapes
          dw      saucer_2
          dw      saucer_3
data_seg  ends

code_grp  group   code_seg                ;linkage for library modules
code_seg  segment para public 'code'
          assume  cs:code_grp, ds:data_seg

          extrn   shape:near
saucer    proc    far
          push    ds                      ;set up FAR return to DOS
          mov     ax,0
          push    ax
          mov     ax,data_seg             ;establish data segment addressing
          mov     ds,ax
          mov     ah,0
          mov     al,4                    ;enter mid-res color graphics mode
          int     10H

          call    draw_saucer             ;draw the saucer before starting
main:                                     ; (so that the first XOR erases it)
          mov     ah,1
          int     16H                     ;see if a key has been pressed
          jnz     s_exit                  ;  if so, exit to DOS
          call    draw_saucer             ;erase it
          call    move_saucer             ;move it
          call    draw_saucer             ;re-draw at new position
          call    delay                   ;let it stay there a while
          jmp     main
s_exit:
          mov     ah,0
          mov     al,3                    ;return to 80x25 text mode
          int     10H
          ret                             ;far return back to DOS
saucer    endp


;----------------------
; DRAW_SAUCER
; Draws the the saucer shape as indexed by the CYCLE value
;  at the coordinates named by CUR_X and CUR_Y
; All registers preserved.

draw_saucer proc near
          push    ax
          push    bx
          push    dx
          push    si
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
          mov     si,offset s_ptrs        ;point to start of 4-element table
          mov     bl,cycle                ;fetch the current state
          mov     bh,0
          shl     bx,1                    ;BX indexes into 4-element S_PTRS table
          mov     si,[si+bx]              ;now DS:SI points to data of shape

          mov     dx,cur_xy               ;fetch both the coordinates
          mov     ah,0                    ;select XOR sub-function
          call    shape                   ;(--- external call draws shape

          pop     si
          pop     dx
          pop     bx
          pop     ax
          ret
draw_saucer endp

;----------------------
; MOVE_SAUCER
; Updates the X,Y, and cycle of the saucer by moving it in
;  the current direction.
; If it touches the edge of the screen, its direction is reversed.
;
; All registers preserved
; Alters the saucer's X,Y and (sometimes) its X and Y vectors

move_saucer proc near

;------- first update current position
          push    dx
          mov     dx,cur_xy               ;fetch current X,Y
          cmp     dl,76                   ;at right side of screen?
          jae     ms_1                    ; yes, change the vector
          cmp     dl,2                    ; no, at left side?
          jae     ms_2                    ;    no, don't change vector
ms_1:
          neg     x_vect                  ;    yes, reverse horiz. direction
ms_2:
          cmp     dh,191                  ;at bottom of screen?
          jae     ms_3                    ; yes, reverse the vector
          cmp     dh,6                    ; no, at top of screen?
          jae     ms_4                    ;    no, don't change
ms_3:
          neg     y_vect                  ;    yes, reverse vert. direction
ms_4:
          add     dl,x_vect               ;update X
          add     dh,y_vect               ;update Y
          mov     cur_xy,dx               ;save new X and Y

;------- now update cycle pointer
          inc     cycle                   ;adjust cycle ptr
          and     cycle,3                 ;cycle 4 back to 0
          pop     dx
          ret
move_saucer endp

;------------
; DELAY
; pauses the action for the count named in variable DELAY_COUNT.
; preserves all registers

delay     proc near
          push    cx
          mov     cx,delay_count
delay_1:  loop    delay_1           ;empty loop
          pop     cx
          ret
delay     endp

code_seg ends
          end     saucer □
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LEGAL

HIGHPOINTS OF SOFTWARE PROTECTION

from Robert Dunaway

When your software goes public or is licensed, be sure you protect your ownership rights.

WHO OWNS THE SOFTWARE?

Be sure you know who owns the software. If you have signed an employment or consulting agreement then someone else may have some claim on the software. Writing programs "after hours" does not guarantee that you are the sole owner. Did you use any company equipment? Did you use information or procedures garnered from work? Is your software a clone of the package you wrote at the office? If any of these apply then you may not own the software outright. Ask an attorney.

You may have "invisible partners" who helped you in some way. Did you use a piece of code from a friend? Did he loan you some equipment? If there is any question, then have them clarify the assistance in writing or give you a bill of sale.

HAVE YOU PROTECTED YOUR INVESTMENT?

Have everyone that sees your special software routines sign a non-disclosure agreement. This puts any future problems in the realm of trade secret infringement if they are definitely unique.

Copyright your software. Once this is done you can take legal action against software pirates and possibly regain the profits that you lost. The application fee is $10. Source code is usually filed. Make sure you include the 'Copyright" notice in all your programs and on the first menu that appears.

EXAMINE THE LICENSE AGREEMENTS CAREFULLY

Your months of hard work can be easily lost by quickly signing a licensing agreement before you have read the fine print. Knowing what other programmers are receiving is important. The best way is to go to someone who is familiar with what a publisher can reasonably be expected to offer. An objective, informed negotiator is very valuable here. Your representative should preferably have previous software contract experience. If that is not possible then an attorney who specializes in art or entertainment law would be the next best choice. Patent and trademark specialist would be good choices too.

MARKETING CONSIDERATIONS

Find a publisher that is extremely interested in your particular package. As the competition for shelf space increases, only an aggressive publisher will survive. You can ally yourself with an agent who will find a buyer for your software. He will take a commission in the neighborhood of 10-15%.

Hardware manufacturers are another source of marketing potential. "Bundling" your software with their hardware is a powerful way to reach a large audience. Your royalty may be much lower, but the sheer volume offsets this. You may even be lucky enough to establish a special agreement with them to write utility software for their hardware.

There are many avenues of marketing your hard work. Check them all out and pick the best one for you and your package.

INTERNATIONAL PC OWNERS

IPCO

"The international user group for the PC"

- Software library of 175+ programs @ $3 each
- Datacount® General Ledger/Manual $35
- Volkswriter® $125
- Bimonthly newsletter 40+ pages
- Worldwide information exchange
- "Freeware"

FOR FREE CATALOG WRITE:

INTERNATIONAL PC OWNERS, INC.
ATTN: FREE CATALOG
P.O. BOX 10426
PITTSBURGH, PA 15234

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DOS CALLS IN PASCAL

from PEEKS n' POKES by Data Base Decisions

DOS functions may be called from Pascal by using the accompanying code. The Pascal routine calls the ASM module, which in turn calls the DOS function. The appropriate registers must be loaded with the necessary parameter information.

```
PROGRAM DOSP(INPUT,OUTPUT);

{demonstrate use of DOS subroutine}

PROCEDURE DOSSUB(VAR INTNO,AH,AL,BH,BL,CH,CL,DH,DL:WORD); EXTERN;

VAR    INTNO,AH,AL,BH,BL,CH,CL,DH,DL:WORD;

BEGIN
       INTNO:=16; {clear screen}
       AH:=6;    {scroll screen up}
       AL:=0;    {blank entire window}
       BH:=7;    {color attribute}
       BL:=0;    {not used}
       CH:=0;    {row of upper left-hand corner}
       CL:=0;    {column of upper left-hand corner}
       DH:=24;   {row of lower right-hand corner}
       DL:=79;   {column of lower right-hand corner}

       DOSSUB(INTNO,AH,AL,BH,BL,CH,CL,DH,DL);      {clear screen}

       INTNO:=16;
       AH:=2;    {home cursor}
       BH:=0;    {page number}
       DH:=0;    {cursor row}
       DL:=0;    {cursor column}

       DOSSUB(INTNO,AH,AL,BH,BL,CH,CL,DH,DL);      {home cursor}

       WRITELN('Screen cleared ...');

       INTNO:=16;
       AH:=2;    {locate cursor}
       BH:=0;    {page number}
       DH:=10;   {cursor row}
       DL:=20;   {cursor column}

       DOSSUB(INTNO,AH,AL,BH,BL,CH,CL,DH,DL);      {locate cursor}

       WRITELN('Cursor positioned at row 11, column 21 ...');

END.
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
; DOSSUB - general dos subroutine interface for compiled BASIC
; CALL DOSSUB(INTNO%,AH%,AL%,BH%,BL%,CH%,CL%,DH%,DL%)
; INTNO% is the interrupt number (0 to 255)
; AH% is the argument corresponding to register AH
; AL% is the argument corresponding to register AL
; BH% is the argument corresponding to register BH
; BL% is the argument corresponding to register BL
; CH% is the argument corresponding to register CH
; CL% is the argument corresponding to register CL
; DH% is the argument corresponding to register DH
; DL% is the argument corresponding to register DL
cseg segment para public 'code'
public dossub
dossub proc far
     assume cs:cseg,ds:nothing,ss:nothing,es:nothing
     push bp
     mov bp,sp
     mov si,[bp+20] ; point to ah argument
     mov ah,[si]
     mov si,[bp+18] ; point to al argument
     mov al,[si]
     mov si,[bp+16] ; point to bh argument
     mov bh,[si]
     mov si,[bp+14] ; point to bl argument
     mov bl,[si]
     mov si,[bp+12] ; point to ch argument
     mov ch,[si]
     mov si,[bp+10] ; point to cl argument
     mov cl,[si]
     mov si,[bp+8]  ; point to dh argument
     mov dh,[si]
     mov si,[bp+6]  ; point to dl argument
     mov dl,[si]
     push ax
     mov si,[bp+22] ; point to interrupt no
     mov al,[si]    ; interrupt in al
     mov cs:intno+1,al ; set interrupt
     pop ax
     push bp        ; save bp - bios bug
intno equ this byte
     int 00h        ; overlaid by intno%
     pop bp         ; restore bp
     mov si,[bp+20] ; point to ah argument
     mov [si],ah
     mov si,[bp+18] ; point to al argument
     mov [si],al
     mov si,[bp+16] ; point to bh argument
     mov [si],bh
     mov si,[bp+14] ; point to bl argument
     mov [si],bl
     mov si,[bp+12] ; point to ch argument
     mov [si],ch
     mov si,[bp+10] ; point to cl argument
     mov [si],cl
     mov si,[bp+8]  ; point to dh argument
     mov [si],dh
     mov si,[bp+6]  ; point to dl argument
     mov [si],dl
     pop bp
     ret 18
dossub endp
cseg ends
     end     □
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EQUATES

DISK PARAMETER EQUATES

by Morton F. Kaplon

I have run across what appears to be a useful aspect of DOS 2.0. Included on diskette with this letter is a file called rmxxx.asm. This is the source code for the RAM DISK (In Storage Diskette (Virtual) with 180K capacity as IBM calls it) device driver starting on page 14-27 of the DOS 2.0 manual. I have modified it as follows.

Right after their MACRO definitions I inserted a new section called

```
;           DISK PARAMETER EQUATES
sctrs_alloc  equu   2     ;Sectors/Allocation Unit;see DOS 2.0 manual C-2
dir_entries  equu   64    ;No. of Directory Entries,same as above
tot_sectors  equu   120   ;Total no. of sectors,same as above
med_descrpt  equu   0fdh  ;Media descriptor,14-22 of DOS manual
add_paras    equu   7500  ;paragraphs to add= (disk size in bytes)/16
no_of_fats   equu   2     ;No. of file allocation tables, see C-2
no_fat_sctrs equu   2     ;No. of sectors occupied by FAT,C-2
```

Elsewhere in the code the above definitions have replaced the actual numbers they used. The code replacements occur on (their line numbers): 88 to 94 inclusive, 111-117 inclusive, and 248. The first two sets are equates for the disk parameters and the last one an equate to set aside memory space for the virtual disk.

Now the nice thing about this is that it turns out to be enormously flexible. You can define any disk size you want. The critical parameters are :

tot__sectors and add__paras.

The structure above has as a base a 360K disk. So to determine the disk size you want you have to change the number of sectors and the amount of memory set aside. The rule is:

tot__sectors = 720*(disk size in Kbytes)/(360)

and

add__paras = (disk size in bytes)/16

The 720 reflects that a 360K disk has 720 sectors, and the divide 16 in the add__paras reflects that addressing aspect of the 8088.

If you put this in your config.sys file and boot you will get a RAM disk with about 56K usable capacity (the difference is the impact of FAT and space for directories), for tot__sectors = 120 and add__paras = 7500.

One can do a similar thing starting as a base with a 180K disk in which case you have to change the med__descrpt to 0fch, and the formula for tot__sectors to

tot__sectors = 360*(disk size in Kbytes)/(180)

There is really no necessity to do this. The 360 here reflects the fact that a 180K disk has 360 sectors.

In my equates above, I have defined the directory entries as 64, which is the smallest number. For a full 360K disk, it should be defined as 112, but the example just shows the flexibility. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BASIC

ARRAY & FILE HANDLING

by Greg Estes

Some years ago I was working on a programming problem and asked a 10 year DP veteran how he would solve it. He listened to my short explanation and said "It all depends." At the time I thought it was a wishy washy answer. He intimated that he would have to know a lot more details to even attempt an answer. The situation had to do with a file layout and whether to put some information in a file or hard code it into the program. There is no simple answer.

Many talented application programmers have achieved their expertise by bearing witness to their own mistakes in previous systems they have designed. If by chance I was given the master key to the first company I did programming for, I would not sneak in on the weekend to rob their safe, no sir. I would quietly make my way into the main computer room and do a global search and replace on their master source library. I would replace "Greg Estes" with "John Q. Doe" in all the source listings. It would remove a great burden from my life.

Our mistakes teach us about file layouts and system design. The longer we have to maintain and enhance a system, the more obvious it becomes how we SHOULD have designed it. But aren't there some rules? Let's see if we can come up with some. Any rule we make can be broken under the right conditions. Designing a system is, in one sense, an attempt to avoid painting yourself into a corner like you've done so many times before. Defensive design, that's the key.

Your coding goes through changes, especially with regard to files. You are perpetually faced with the question of where to put pieces of information. As your skill increases you find that you have passed through three levels of enlightenment.

Level One : Put EVERYTHING in an ARRAY
Level Two : Put EVERYTHING in a File
Level Three : Put things in Files or Arrays (it all depends)

Usually you start by stuffing tons of information in arrays. It is convenient. You soon find that the program just BARELY fits into 64K. Imagine what is going to happen 2 months down the road when you have to add more code. Bad news. As long as you are in a ONE PROGRAM - ONE FILE situation this may be OK. When you find yourself with more that one program and several files, then you are setting yourself up for some heavy maintenance down the road.

For example: you are asked to put together a small system for a clothing manufacturer. They have 10 department codes. Should you store the codes in the program in an array?

```
100 A$(1) = " ACT"
110 A$(2) = " ADM"
120 A$(3) = " INV"
...
```

Or put them in a file? If they have used the same department codes for 10 years (and can prove it to me by producing an old report) I might do it. But knowing human nature, I bet that a new president will come along and add a new department. The extenuating circumstances are that department codes normally have descriptions with them, and I know that companies like to play around with those. So I would put them in a file. Even though there are just 10.

Maybe the rule is: if it is changeable at all, put it in a file. Once you find yourself creating quite a few tiny files, and you do not want to hassle with so many, just create a single file with an identifying field. Instead of a separate file for department codes, job descriptions, and room descriptions, you would make a file that resembles this:

(D = DEPT RECORD, J = JOB RECORD, R = ROOM DESCRIPTION)

```
D ACT ACCOUNTING DEPARTMENT
D INV INVENTORY DEPARTMENT
D PAY PAYROLL DEPARTMENT
J JAN JANITOR
J SEC SECRETARY
R 25  COMPUTER ROOM
R 13  ADMINISTRATIVE OFFICES
R 123 KITCHEN
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

When one program wants the department codes, open the file and read the "D" records into an array. The file structure above is a solution to the problem of lots of little files with not enough space to open all of them in one program. Maybe the rule is: when things start to get really bad, try and clean things up. Consolidate.

When DO you decide to hard code an array with information in a program? It is OK when the information does not change; days of the week, calendar months, a table of fixed values. Arrays are for speeding up the execution time of a program. If there are 10 department codes, store them in a file, read them into an array, and process them quickly. An array is a vehicle for speed, not for holding dynamic information.

There is a rule that every software writer knows. LEAVE SOME EXTRA SPACE. Leave it everywhere; in an array, in a file, on a diskette. If a company has 10 department codes, what should you DIM the array size to be? Well 15 or 20 is a good choice. If they have 40 job descriptions, use an array size of 60 or 70. When you design a file, leave an extra field at the end of the record layout with 20 or 30 spaces in it. Somebody is going to want to add a new field or new code in there. Be sure and initialize the field with spaces when you create, read, and write it. The future addition of fields can then occur in an organized fashion; you know what is out there.

Files are often optimized by using records with a size that evenly divides into 512. Use a record size of 32 or 64 or 128. Leave some room at the end of the file layout.

As you are building your program, allowing extra space in arrays and files, you will notice the code size beginning to max out. The extra room you incorporated sends early warning signals.

Perhaps you are trying to accomplish too much in one program. Time to make it two programs. The "worst case" scenario is painstakingly building a lean and hunger program with no extra space anywhere, finally getting it to barely fit into 64K. One month down the road you need to add some code that won't fit. Welcome to a MAJOR overhaul on several programs as well as your initial system design. I can't think of a better way to destroy a custom programming budget. □

SHORT KEYBOARD CORD

This simple solution to the problem of a short connection cord between the IBM PC chassis and the keyboard has saved me a lot of back aches. Here it is. Starting with the end of the cable that attaches to the IBM PC chassis, uncoil 4 coils so they form a straight wire. Plug the cable into the IBM PC and slide the wire under the chassis. If the straight portion of the wire is not long enough, unwind another coil. Note: do not just stretch the coils, actually uncoil them. When you are finished, you should be able to sit back in your chair with the keyboard in your lap. I put in 10 hours a day, 6 days a week and find my back doesn't bother me as much.

Elmer W. Fittery
International Computers
Mexico Beach, FL

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NIKE MARATHON RACE RESULTS WITH dBASE II

Programs from Jim Boble

The following two programs are part of a dBASE II system that keeps track of the results from a marathon race. In this case the NIKE marathon.

As the marathon runners cross the finish line, their finish times are recorded and entered into a BATCH results (BATCHR) file. Only their race number and finishing time is entered. Once the BATCHR file is of sufficient size, all of the finishing times are posted to a final RESULTS file. The program that handles this is RPOST.PRG. Notice that the race numbers are numeric in the BATCHR file; this is to facilitate error checking. But in the RESULTS file the race numbers are character fields, to make sorting and indexing easier. Take a look at the conversion of a numeric field to a character field, adding single quotes in the process, so that the field can be used in a FIND statement. Leading spaces are not handled well if you need to use FIND.

There is a master file that holds the complete statistics of a runner, called MASTERF.DBF. After the race is over and all of the finishing times have been posted to the RESULTS file, the runners and officials want a complete printout. RFALPHA.PRG reports All Finishers Alphabetically. It reads the RESULTS file and the MASTERF file to create the comprehensive NIKE report. With the results and the runner information in separate files, you have maximized the ability to produce whatever reports are desired. One report is by finishing times. Another report is by last names. Still another is by age group.

```
*  RPOST.PRG  Posts Batch Results to Master Results Database          09/10/83
*     The databases BATCHR and RESULTS each contain the following fields:
*     RACE:NUM  and  FIN:TIME.

*     The object of RPOST is to update the FIN:TIME in the RESULTS database
*     from the FIN:TIME in the BATCHR file for each corresponding (same
*     RACE:NUM) record.  This is accomplished by sequentially reading records
*     in BATCHR, finding the corresponding record in RESULTS and replacing the
*     FIN:TIME in RESULTS with that in BATCHR.  Note that BATCHR is not indexed,
*     while RESULTS is indexed by RACE:NUM.

*  clear the screen and set defaults
erase
set colon off
set bell off
set talk off

*  initialize variables
store '                                                              ' to PLINE
store '      ' to RNUM
store '     '  to ACT:TIME
store 0 to TOT:POST
store 'N' to VERIFY

*  declare BATCHR as primary database, RESULTS as secondary database
*  declare RESULTS to be indexed by RACE:NUM using RACENR index file
*  select BATCHR file and goto top / first record
use BATCHR
select secondary
use B:RESULTS index B:RACENR
select primary

goto top

*  read the BATCHR database sequentially until reaching end of file
do while (.not. eof)

*  form RACE:NUM key from BATCHR
   store "'" + str(RACE:NUM,4) + "'" to RNUM
*  select RESULTS database, find record with RACE:NUM
   select secondary
   find &RNUM
   store str(P.FIN:TIME,5) to ACT:TIME

*  check to see if a record with RACE:NUM as key is found
*  if a record is 'not' found, # will equal zero.  if so print error message.
   if # = 0
     set print on
     store 'INVALID RACE #:' + str(RACE:NUM,4) to PLINE
     store PLINE + '    ' + $(ACT:TIME,1,1)+':'+$(ACT:TIME,2,2)+':' to PLINE
     store PLINE + $(ACT:TIME,4,2) to PLINE
     ? PLINE
     set print off
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
*  if a corresponding record is found, then first check to see if a FIN:TIME
*  has already been posted for the RACE:NUM.  if so print error message
   else
     if S.FIN:TIME)'      '
        set print on
        store 'ALREADY POSTED:' + str(RACE:NUM,4) to PLINE
        store PLINE + '    ' + $(ACT:TIME,1,1)+':'+$(ACT:TIME,2,2)+':' to PLINE
        store PLINE + $(ACT:TIME,4,2) to PLINE
        ? PLINE
        set print off

*  all error conditions are negative -- so update (via replace) FIN:TIME
*  in RESULTS.  increment the number of records posted, thus far.
     else
        replace S.FIN:TIME with ACT:TIME
        store TOT:POST + 1 to TOT:POST
     endif
   endif

*  flip back to the primary BATCHR database and skip to (read) next record
   select primary
   skip

enddo

*  report the number of records posted to RESULTS
set print on
?
?
? 'Records Posted:' + str(tot:post,3)
?
set print off

*  perform final housekeeping and clean up
set talk on
use
release all
```

```
*  RFALPHA.PRG  Report of All Finishers Alphabetically              09/10/83
*    This procedure uses two databases to produce an alphabetical listing
*    of race finishers.  The two databases, MASTERF and RESULTS, are each
*    indexed.  The MASTERF database is accessed by an alphabetic ordered
*    index on LASTNAME and FIRSTNAME.  The RESULTS file is accessed by an
*    index based on RACE:NUM.  The procedure reads through the MASTERF database
*    in alphabetic order and uses the RACE:NUM to lookup the corresponding
*    record in the RESULTS database.  A report line is written using in-
*    formation from each database.

*  set standard parameters
set colon off
set bell off
set talk off

*  initialize variables
store '                                                    ' to PLINE
store '        ' to TIMESTR
store '       ' to RNUM
store 'P' to OUT:TYPE
store 100 to LINESP
store 'N' to VERIFY

*  sub-procedure to identify report, prompt for destination of report and
*  verify choice.  repeat prompt until satisfied.
do while (VERIFY='N')
   erase
   @ 00,00 say 'Alphabetical Finisher's List'
   ?
   accept 'Report to (P)rinter or (F)ile? ' to OUT:TYPE
   store !($(OUT:TYPE,1,1)) to OUT:TYPE
   ?
   accept 'Verify (Y or N) ' to VERIFY
   store !($(VERIFY,1,1)) to VERIFY
enddo

*  declare MASTERF as primary database and set alphabetic index
*  declare RESULTS as secondary database and set race number index
*  position at beginning of MASTERF database
use B:MASTERF index B:ALPHA
select secondary
use B:RESULTS index B:RACENR
select primary

goto top
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
*  set up printer or disk file for report depending on option selected
if OUT:TYPE='P'
   set print on
else
   set alternate to B:RFALPHA.PRT
   set alternate on
endif
set console off

* read records from the MASTERF database until reaching the end of file
do while (.not. eof)




*  form key from RACE:NUM, select RESULTS (secondary) database and find record
 store "'" + RACE:NUM + "'" to RNUM
 select secondary
 find &RNUM

*  check for new page and if so, print title and report headings
 if LINESP )= 50
   ? chr(12)
   store '                          1983 Wild Goose Chase Fun Run' to PLINE
   ? PLINE
   ?
   store '                                Official Results' to PLINE
   ? PLINE
   ?
   store 'Place   Race #   First Name     Last Name          ' to PLINE
   store PLINE + 'City        Sex Age   FinishTime' to PLINE
   ? PLINE
   store '-----   ------   ------------   ----------------   ' to PLINE
   store PLINE + '---------   -------   ----------' to PLINE
   ? PLINE

   store 0 to linesp
 endif

* check to see if record can be found in the RESULTS database and if there
* is FIN:TIME recorded.  if so, form report line and print (or send to disk).
 if FIN:TIME()'      ' .and. #()0
      store str(PL:OVER,5) to PLINE
      store PLINE + '    ' + RACE:NUM to PLINE
      store PLINE + '    ' + FIRSTNAME + '    ' + LASTNAME to PLINE
      store PLINE + '    ' + CITY to PLINE
      store PLINE + '     ' + SEX +  '- ' + str(AGE,3) to PLINE
      store PLINE + '     ' to PLINE
      store $(FIN:TIME,1,1) + ':' + $(FIN:TIME,2,2) + ':' to TIMESTR
      store TIMESTR + $(FIN:TIME,4,2) to TIMESTR
      store PLINE + TIMESTR to PLINE
      ? PLINE
      store LINESP + 1 to LINESP

 endif

*  flip back to the MASTERF database and skip to (read) the next record
*  which will be the next record in alphabetic order
   select primary
   skip

enddo

*  clean up before exit
?
if OUT:TYPE='P'
   set print off
else
   set alternate off
endif

*  reset standard parameters
set console on
use
release all
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MAINTAINING YOUR COMPILER LIBRARIES

POLYLIBRARIAN ($99)
Polytron
PO Box 787
Hillsboro, OR 97123
ph. 800-547-4000

Full featured compilers have large libraries that come with them. Normally you use only a small portion of the functions and suffer the agony of having a huge file on your diskette taking up space. If you write your own ASM modules, and want to LINK them together, you can easily end up with long command lines that are no fun either.

What you really need is a library maintenance program. POLYLIBRARIAN is designed solely for that purpose. It is an extensive, friendly utility to add, change, delete, backup, extract, and replace the OBJ modules in your library. The program has three modes of operation: interactive, command file, and command line. Programmer's utilities programs are often difficult to use with cryptic documentation. POLYLIBRARIAN is well polished in both areas.

Why not go to work on a library and see just what it is like? You enter the program by typing "LIB." You are then in the interactive (interpretive) mode. We will look at the BASCOM.LIB supplied with IBM's Basic Compiler. Once in the object module librarian program you need to select a library:

```
*S BASCOM
```

The program creates a large temporary work file to allow changes. To list the modules in BASCOM type:

```
*L
```

Here is a partial list of the functions IBM provides.

```
Listing of modules in library - C:BASCOM.LIB
   Number    Size  Opt.   Name
        1     367H  1111  ADVGRP
        2     25BH  1111  ATAN
        3     2DAH  1111  BLOAD
        4     5F9H  1111  CIRCLE
        5     522H  1111  CONASC
        6     2BEH  1111  CONST
        7     409H  1111  CONV
        8     780H  1111  DMATH
        9     309H  1111  DOSDAT
       10     1E9H  1111  DOSINI
       11     3E6H  1111  DOSRUN
       12     3C6H  1111  DRAWML
       13     2A2H  1111  DATAN
       14     315H  1111  DEXP
       15     2E3H  1111  DLOG
       16     305H  1111  DPWR
       17     2F7H  1111  DSIN    ....
```

You may also look at the PUBLICS.

```
*L P


BLOAD
  $BL0      $BL1      $BSV

CIRCLE
  $CI0      $CI1      $CI2      $CI3

CONST
  $ARG      $DBL4TH   $DBLHALF  $DHALFPI  $DHALFSQR2
  $DLN2     $DLOG2E   $DRECIP2PI          $DSIXTHPI
  $DSQRT3   $DTANTWELFTHPI      $HALFPI   $HALFSQR2
  $LN2      $LOG2E    $ONE4TH   $ONEHALF  $PUT1
  $PUT1D    $RECIP2PI           $SIXTHPI  $SQRT3
  $TANTWELFTHPI       $TEMP     $TEMP2

CONV
  $ADC      $ADCA     $ADCB     $C8IA     $C8IB
  $C8IC     $C8ID     $CDPA     $CDPB     $CIDA
  $CINA     $CINB     $CINC     $CIND     $CINDP
  $CINFAC   $CINSP    $CISA     $CSPA     $CSPB
  $V8IA

DMATH
  PMULD     $DADD     $DDIV     $DMUL     $DNORM
  $DROUND   $DSUB     $FADB     $FADD     $FADF
  $FADH     $FDVB     $FDVD     $FDVF     $FDVH
  $FMUB     $FMUD     $FMUF     $FMUH     $FSUB
  $FSUD     $FSUF     $FSUH

DOSRUN
  $CHN      $RUN
```

Look through the modules and decide which ones you may never need, then just delete them. Extracting a module is just as simple. POLY will remove it from the large library and put it in another one.

Think of the possibilities. You can extract an OBJ module, change it, and put it back in. You can replace a module with your own. You can create a number of special functions that you use a lot and add them to the main library. You add and change modules on a regular basis during development, POLY offers you a command file mode that allows access to all the functions from a text file rather than the keyboard, really just a batch file approach if you do frequent updates. Referring to modules using wildcards is also possible. You may change internal module names, hide PUBLIC names, and rearrange the order of the modules. Online HELP is provided if you forget any commands.

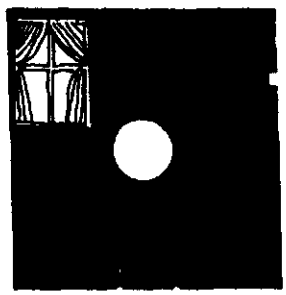
The POLYLIBRARIAN handles OBJ files in Intel's standard Object Module Format (OMF-86). This format is described in a publication called "8086 Relocatable Object Module Formats." Ask Intel for a copy of booklet #121748-001. Polytron provides a brief summary of the format in their manual: record header types, link information records, data records, etc.

The Object Module Librarian from Polytron is an excellent tool for serious programmers. It combines professional quality with a flexible user interface. / PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SOFTWARE REVIEWS

THE dBASE II WINDOW



Reviewed by Jim Bohle

The dBASE II WINDOW
Tylog Systems, Inc.
9805 SW 152 Terrace
Miami, Florida 33157
(303) 253-5942

In addition to word processing and spreadsheet software one of the most significant and useful application tools for computer users today is the relational database system. Simply put, database systems perform the laborious tasks of describing, storing, organizing, retrieving and reporting information. They are designed to facilitate ease in defining and implementing the user's application and eliminate the need for the user to solve his problems in some general purpose language.

While several database systems are available for small computers, the most widely used product is dBASE II from Ashton-Tate. dBASE II is a relational database system with an extensive set of commands and options supported by a richly-structured programming language. The language is used chiefly to automate the standard sequences of commands used in interacting with the database. However, dBASE II critics suggest that users cannot implement applications without becoming dBASE II programmers. Some feel the user-friendliness of dBASE II is tied to one's ability to use the dBASE II programming language.

Just as there have been several attempts to ease the burden of programming in many of the common computer languages such as BASIC, PASCAL and COBOL, a new market of products aimed at generating dBASE II application programs has emerged. Approximately half a dozen such systems are currently available. Typically, the dBASE II program generators allow a user to easily describe the database structure, application menus, database maintenance procedures, entry/query screens as well as report layouts. One such product is The dBASE II WINDOW offered by Tylog Systems, Inc. of Miami, Florida.

PRODUCT OVERVIEW

The dBASE II WINDOW (referred to as the WINDOW) is substantially more than a dBASE II program generator. While there are several products which provide the aforementioned facilities of database description, application menu generation and so on, most say 'bon voyage' when it comes to actually developing and using the application at hand. The user ends up with a database and a collection of programs which may or may not serve the user's needs. The WINDOW attempts to be a complete resource in the application development process. The emphasis is on training the user to properly and thoroughly analyze, define and generate a dBASE II implementation of his application needs. The system is especially useful in applications involving relationships between multiple databases with varying screen and query requirements. After all, dBASE II is designed as a relational data base system. As with most program generators, the WINDOW permits the user to describe his desires in a convenient and natural fashion. The WINDOW automatically translates these requests into a dBASE II program.

The WINDOW is largely menu-driven. It is divided into two sub-menus: data access functions and database utility functions. The data access functions implement the application by allowing database records to be added, changed, deleted, queried, and reported. What is significant is that WINDOW has actually placed most of the dBASE II commands in a convenient screen-menu format. In this sense, the WINDOW is a menu-driven implementation of the dBASE II system commands. The data access menu includes an extensive report generator which permits reports to be drawn from multiple databases. The utility functions provide the means to define, create, modify and document applications. This sub-menu guides the user through the design and implementation of databases, screens, and reports.

DOCUMENTATION

The documentation for the WINDOW consists of an easy to read 118 page manual. The amount of attention given to discussing the general concepts of databases, relations, application development and dBASE II specifically is impressive. The user can learn a great deal about the nature and utility of relational databases from this manual.

The manual is divided into three sections: overview and operation of the WINDOW, the database application development process and a reference section on WINDOW specifics. The introduction to databases and related concepts is informative but should be developed a bit more gradually — some of the concepts presented in the introductory examples on relational databases are a bit complex. The operational section expands the example applications quite well and adequately describe the capabilities of the WINDOW. The section dealing with the database application process describes the central issues of how to analyze, define and generate a database application using the WINDOW. Sample forms for database definition and screen layouts are given in the reference section of the manual which also includes installation instructions and implementation details.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OPERATION

The diskette on which the WINDOW is distributed contains 63 files which includes 39 operational dBASE II programs and database, index and special screen files. This is the working set of files for operation of the WINDOW. The remaining files are demonstration files for their sample applications. Please note that the diskette does NOT contain the dBASE II programs or an operating system. Operation of the system is started by installing the WINDOW. This is quite simple and runs as documented in the manual by issuing the command: dBASE WINSTALL. Installation gives the user the chance to specify the default and number of disk drives as well as some standard dBASE II screen data entry parameters.

Once installed, the system is invoked by issuing the command: dBASE WINDOW. A system date screen is presented followed by the WINDOW master menu which presents the user with the choice to:

Access the Data
Create or Modify Databases, Screens or Reports

It should be noted that every menu provides a Help facility which is first-rate. Not only are the Help screens informative but they page correctly and even inform the user of how many pages of Help follow.

In selecting the second option to Create/Modify Database, Screens or Reports, the user is presented with the utility sub-menu options to:

Create or Modify Screen or Report Formats
Document Screen or Report Formats
Delete Screen or Report Formats
Delete Databases
Pack Databases
Change Today's Date

Databases and their associated indexes are defined and created as part of the screen creation / modification process. A nice feature of the first option is that multiple screens may be created for a single database. That is, a database need not be created as part of the process of creating a screen. Some screens may be used for data entry/update while others might be used to update selected fields or for record query purposes. The documentation option essentially provides the capability to print screen and report formats — a very useful feature. The deletion and packing of databases is a direct implementation from dBASE II.

The data access sub-menu provides the user with the ability to:

Update (Add, Change, Search or Query Records)
Report on Data
Global Operations
Name the Relation to Use

Usually the first option selected is to Name the Relation to Use. This is important since the entire system is oriented around defining and using relations among databases. Relations may define the index or access order on a single database or the access link between two databases. They are defined by the user and maintained by the WINDOW in a relation catalog database — a dBASE II database itself. The current relation in use is conveniently displayed on the data access sub-menu. Once a relation is identified, the user may enter the Update, Report or Global Operations modes. The update option provides for a myriad of database functions — essentially all the common operations available in dBASE II directly. The WINDOW provides a more user-friendly, menu-driven product then dBASE II offers on its own. The functions make use of the user-defined update and query screens. The Report on Data option allows the user to invoke any of the defined database reports. Global Operations include deleting or recalling records, changing records (ala the dBASE II Replace and Update commands), browsing records and importing and exporting data from and to external files.

CONCLUSIONS AND COMMENTS

The WINDOW provides the dBASE II user with far more than a program code generator. It is essentially a relational database application generator coupled with a menu-driven implementation of dBASE II. Novice database users can gain a great deal of information about how to analyze, define and properly implement relational database applications. The documentation and on-line help facility are first-rate. The real life sample applications included are presented well in the documentation.

The product does have some drawbacks. Performance of the system for the most part is 'Sluggish'. First, the interpretive nature of dBASE II does not lend itself to quick running application programs and the WINDOW is, of course, implemented in the dBASE II language. Secondly, there appears to be a good deal of overhead in maintaining the relations catalog, indexes and in generating application programs. The processes of creating screens, reports and databases is painstakingly slow. When combined with an operating system and dBASE II , the amount of disk storage required by the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WINDOW is quite large — only 20K bytes remained on a standard 360K (DOS 2.0) diskette. A system configured with a 10MB hard disk would improve the storage and execution speed constraints. There were a few operational problems like not being able to import data from an external file — the disk thrashed for 15 minutes without adding any records to the newly created database. The editor used for creating screens, reports and programs is largely that employed by dBASE II and is so cumbersome that it is nearly unusable.

While the window is probably an excellent tool for the non-programming database user, it may remove too much control from the more sophisticated dBASE II user. For example, the limitation of five indexes for a single database is questionable and the use of non-standard file extensions for some files is unfortunate. A real dBASE II hacker may want more control over the implementation of his applications. In any case, the WINDOW is a product which merits close attention if one is serious about implementing database applications in dBASE II. Its operation is generally reliable and the $249 price tag is very reasonable for this type of product.

Comments from Rich Goldner of Tylog: The WINDOW uses approx. 188K of disk space. dBASE II uses 70K. Importing files in a standard format works well. The editor mentioned in WINDOW is, of course, the same editor that dBASE II provides. Index files can be given any name. We use something other that .NDX for our names, but it makes no difference to dBASE II what they are called. Whenever you use both a primary and secondary file, you need to remember that dBASE II sets a limit on the number of files you can open. You have 2 files opened right away with the primary and secondary ones, then you are allowed to open 5 indexes □

PJ LIBRARY

ULTIMATE PC INTERFACE BOOK

If you need to know the inside out story on the IBM PC communication hardware, ports, chips, programming considerations, then you should pick up this book fast, it may be gone by the time you make it to the store. The title is "Interfacing to the IBM PC." The writer is none other then the "former" head of the PC hardware design section. The book is for hardware hackers, no question about it. If you are frustrating yourself with I/O hardware problems, the answers you need are contained in this one. Another quality book from Howard W. Sams.

INTERFACING THE IBM PC by Lewis C. Eggbrecht
Howard W. Sams & Co., Inc.
4300 West 62nd St.
PO Box 7092
Indianapolis, IN 46206
ph. 800-428-3696

USEFUL PHRASES

"I'd really like to sit down and talk with you, but I have o get back home, I left my compiler running."

Real Programmer
Silicon Valley, CA

PC ABSTRACTS

Finally we have a cross reference and publication index for the entire PC line of publications. PC ABSTRACTS will be a comprehensive listing of any article that deals with the IBM PC. The articles will be summarized so you can get an idea of what they discuss. The 1983 addition will be out shortly after the first of the year.

PC ABSTRACTS
9754 S. Houston Ave.
Jenks, OK 74037

APPLE TO IBM CONVERSIONS

Blue Water Systems has written a booklet illustrating some of the major points in converting Applesoft programs to IBM PC BASIC. The booklet is straightforward and not overly technical. If you are new to conversions it would be helpful.

Applesoft to IBM PC
Blue Water Systems
PO Box 1877
Oceanside, CA 92054

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MISC.

C PROGRAMMERS

The are a couple of BBS's that deal with C topics. A main one in New Jersey called the "C LINE" revolves around UNIX and C implementations with some definite interest in IBM PC implementations. To access the C Line call (201)-625-1797. One other bulletin board in Chicago has some C library routines available for Lattice versions. Good luck on finding this BBS not being used. It is provided by Gene Plantz and the number is (312)-882-4227, the NW INFO EXCHANGE.

MODULA-2 COMPILER

Modula-2 has been released for the IBM PC. The compiler is reasonably priced, $40. You can even purchase the source code for $160. The organization that supplies it is non-profit.

The Modula Research Institute
Provo, UT
ph. 801-375-7402

SPREADSHEET FORMULA FORMATTER

Here is an new debugging tool for large spreadsheet users. It is an inexpensive utility to print out the FORMULAS of your spreadsheet. You can display complete cell formulas, constants and formatting attributes. If you have been swimming around in someone else's enormous financial module looking for that circular bug, this package will give it to you in hard copy.

FORMULA FORMATTER ($25)
Decision Technology
20760-I 4th St.
Saratoga, CA 95070
ph. 408-867-6298

IBM STYLE PLASTIC BOOKLET BINDERS

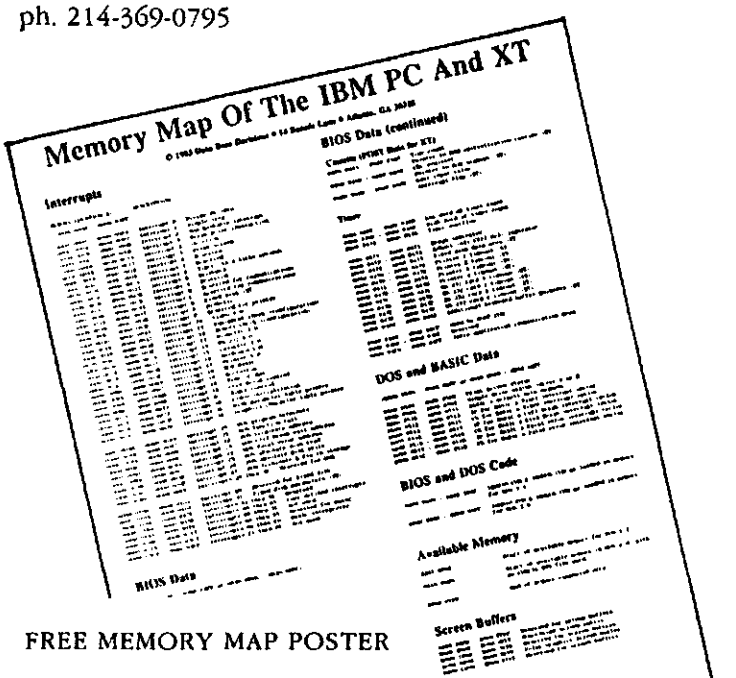
The same plastic binders that many IBM products come in can be purchased from:

Communications Packaging Corp.
2635 South Santa Fe Dr.
Denver, CO 80223
ph. 303-758-1113

HEBREW WORD PROCESSOR

TexaSoft has announced the first word processor in the United States that supports the Hebrew language - VERSATEXT. It is a full featured word processor that has been out for 2 years and how supports the Hebrew language. You can switch from Hebrew back to English in one key stroke. The package includes a chip to generate Hebrew characters, as well as another one for the Epson printer.

Hebrew VersaText
TexaSoft
One Energy Square, Suite 660
Dallas, TX 75206
ph. 214-369-0795



FREE MEMORY MAP POSTER

Part of my latest software package "The Inside Track" includes a memory map for the IBM PC and XT. The fold-out chart shows the important memory locations of the PC in one place, rather than being scattered through the DOS, Technical Reference, and other manuals.

I will mail a FREE copy of the memory map poster to any PJ readers who send me a self-addressed stamped envelope before April 1, 1984. The envelope should be the standard business size (#10) or larger.

Brett Salter/Memory Poster
Data Base Decisions
14 Bonnie Lane
Atlanta, GA 30328
ph. 404-256-3860

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WE KNOW YOU'RE SERIOUS ABOUT THE IBM PC.

SO ARE WE!

WE KNOW WHAT YOU WANT.

- *Detailed how-to articles*
- *Programming tips*
- *In-Depth product reviews*
- *Problem reporting and solutions*
- *Code listings*

WE'VE ALWAYS HAD IT ...

Some features in previous issues were a technical look at the system unit, number crunching, utility programs, asynchronous communications, monitor and printer character sets, personalizing DOS, BASIC vs. Pascal, menu generation programs, unleashing your printer, evaluation of hard disk drives, and plotting data with PC graphics.

WE ALWAYS WILL.

Features for 1983 will include extensive information about networks, database managers, spreadsheets, operating systems, the 8087 chip, professional program development, and much more.

WE KNOW WHICH MAGAZINE YOU NEED ...

PERSONAL COMPUTER AGE

The Definitive Journal for the IBM Personal Computer User

CALL TOLL FREE: 800-824-7888 operator 77
California only: 800-852-7777 op. 77

I NEED WHAT YOU'VE GOT! ENTER MY SUBSCRIPTION TODAY!

☐ I WANT INFORMATION ABOUT BACK ISSUES, TOO!

☐ PAYMENT ENCLOSED ☐ BILL ME

CHARGE MY:

☐ VISA CARD NO. _____

☐ MASTERCARD EXPIRATION DATE _____

AUTHORIZED SIGNATURE _____

☐ 1 yr. (12 issues) $24.00
☐ 2 yrs. (24 issues) $45.00
☐ 3 yrs. (36 issues) $64.00
☐ NEW ☐ RENEWAL

FOREIGN SUBSCRIPTIONS
1 YR. (12 ISSUES)
CANADA/MEXICO $29.00

OTHER COUNTRIES,
WRITE FOR RATES
PAYMENTS MUST BE
IN U.S. FUNDS DRAWN
ON A U.S. BANK

NAME _____

ADDRESS _____

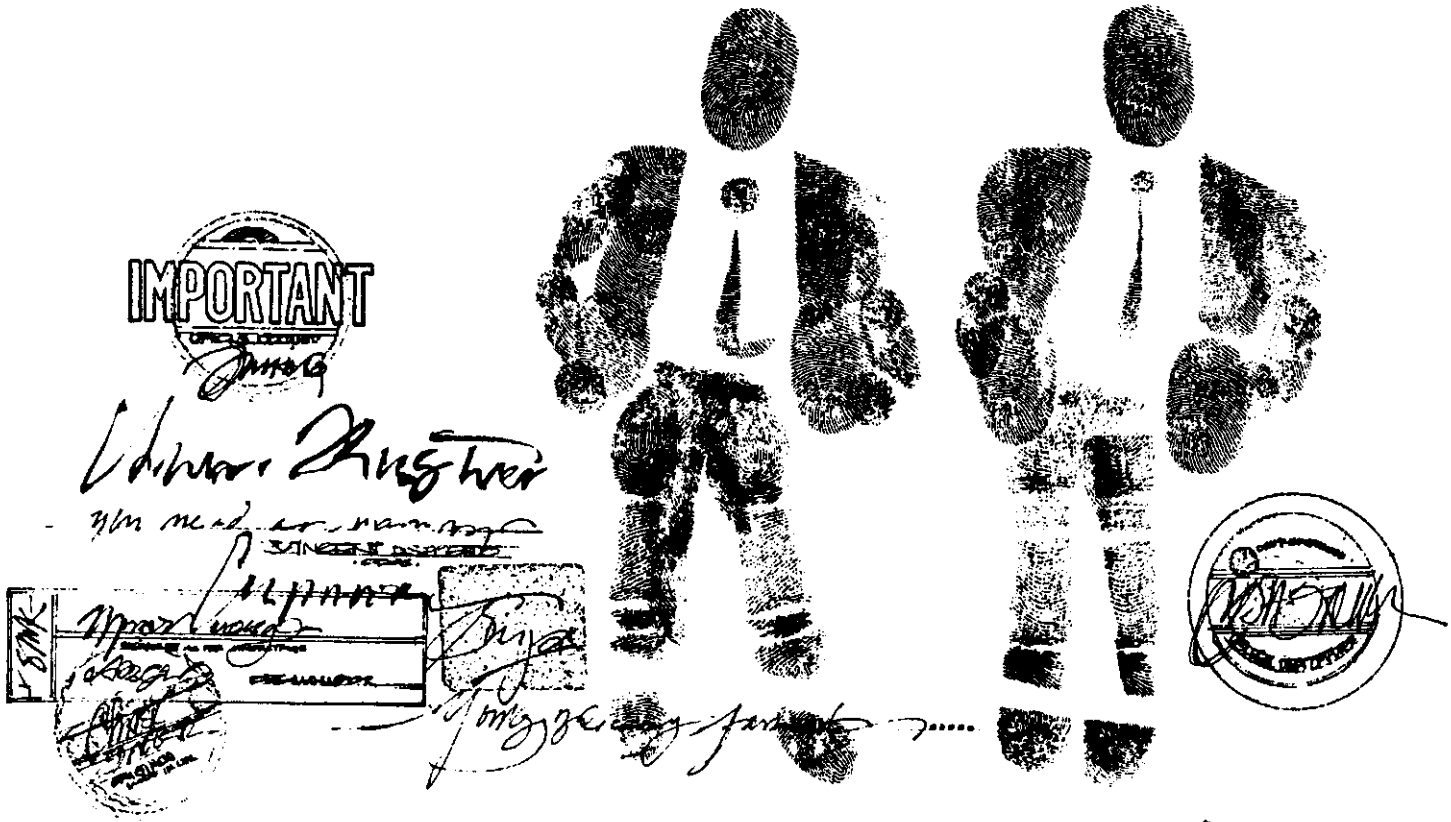
CITY _____

STATE _____ ZIP _____

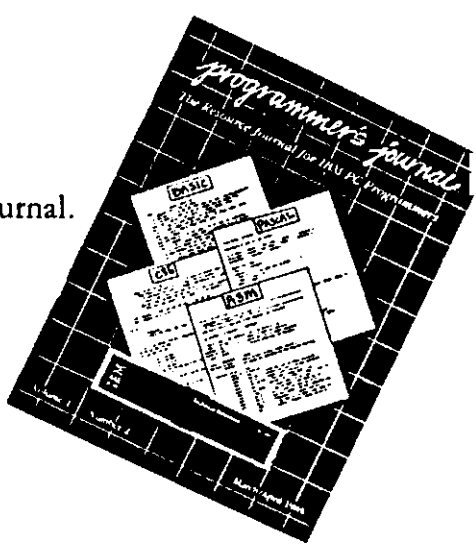
Personal Computer Age

Department PJ5
P.O. Box 70725
Pasadena,
California 91107

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



You can reinvent the wheel or you can subscribe to the Programmer's Journal.



Vol. 1 #1: Norton Pascal I/O, Program style manual, COBOL, System Analysis, BASIC modules, Rounding error.
Vol. 1 #2: Defining character sets, C86 BIOS calls, Copy protection, Pascal code, Input routines, Writing an Assembler.
Vol. 1 #3: ASM dumb terminal, C source code, ASM graphics line drawing, Text editors reviewed, utilities.
Vol. 1 #4: Diskette directory via Pascal, MASM WHILE-END, C/ASM interface, ASM stay resident routine, structured programming.

SUBSCRIPTION

Include your:

Name _____

Company _____

Address _____

City _____ State _____

Zip _____

Languages you use: _____

Languages interested in: _____

1 year bi-monthly subscription rate:
$24 U.S. ($30 Canada, $40 other)
☐ Check enclosed (all back issues sent)
☐ Bill me

Begin my subscription with:
☐ First issue
☐ Current issue

Send to:
PROGRAMMER'S JOURNAL
2765 Potter St.
Eugene, OR 97405
Ph. (503) 687-0398

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ZAVENTEM

Dr. Robert Rosner
CENTER FOR ASTROPHYSICS
60 GARDEN STREET
CAMBRIDGE, MASSACHU

AIR MAI
AVIO

K. G. Berge, M.D.
Mayo Clinic
Rochester, Minnesota 55905

Dun & Bradstreet

a company of
The Dun & Bradstreet Corporation
East Jefferson Street, Rockville, Md. 20852

DAYTON, OH 454
PM
1985

The Great S
of the United Sta
1782-1982

USA 20c

DR. DHIRAPHORN SRIFUENGFUNG
บริษัท ไทยอาซาฮีโซดาไฟ จำกัด
THAI ASAHI CAUSTIC SODA CO.,LTD.
THASCO
2nd FLOOR, CATHAY TRUST BUILDING,
TEL. 233-1231-8

Atomic Energy of Canada Limited
Engineering Company
Sheridan Park Research Community
Mississauga, Ontario L5K 1B2

L'Energie Atomique du Canada, Limitée
Société d'Ingénierie

Rosby

AIR MAIL
CANADA
32

PEAT MARWICK

FAX BOX - MCD
Peat, Marwick, Mitchell & Co.
Certified Public Acco ts
345 Park

MODULAR COMPUTER SYSTEMS
10 Cosdown Close, Tullamarine, Victoria, 3043, Australia

TULLAMARINE
VICTORIA
R No 1724

AR.

PROGRAMMER'S JOURNAL
2765 POTTER ST,
EUGENE, OR 97405
U.S.A.

$4.00

AUSTRALIA
$2
FRED McCUBBIN On the Wallaby Track 1896
Art Gallery NS

Australia 25c

AUSTRALIA
FRED McCUBBIN On the Wallaby Track 1896
Art Gallery NS

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED